TÉCNICO LISBOA

nuBOINC - BOINC para as massas

Patrick Johann Pircher

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor(s): Professor João Nuno de Oliveira e Silva
Professor João Coelho Garcia

Examination Committee

Chairperson: Prof. Nuno Cavaco Gomes Horta
Supervisor: Prof. João Nuno de Oliveira e Silva
Member of the Committee: Prof. Luís Manuel Antunes Veiga

November 2014

B

Make everything as simple as possible, but not simpler.

Albert Einstein

Acknowledgments

First of all, I would like to express my gratitude to Prof. João Nuno Silva, who tutored this work, for his suggestions and remarks. His support, patience and dedication made this work easier to develop.

I would also like to thank my family for their support as well as my brother and Ms. Abery for helping and supporting me on this thesis.

Last, but not least, a big thank to all my friends and colleagues.

Abstract

At the present time, it is difficult for users to benefit from computer cycle sharing over the Internet, even with well known architectures such as BOINC, since it is difficult for an ordinary user to install the required infrastructure, develop the processing applications and gather enough computer cycle donors. In general, computer owners only have one role in the system: to donate their computers processing power.

In this thesis we describe a set of BOINC extensions that allow any user to create and submit jobs that can take advantage of remote idle cycles. In order to submit their jobs, users only have to provide the input files, define the processing application as well as the arguments that will be provided to the application. Later, other users will contact the server, receive a set of jobs, and process them in a secure virtual machine. Consequently, all jobs will be processed inside an isolated environment, unable to compromise the processing computer. This is required, since users can submit any type of application.

The virtual machine and all required files for the job are transferred using the BitTorrent protocol. The files must be hosted by the project creator. Similarly, all generated output files will be sent directly to the project creator, thus minimizing the files saved on the BOINC server.

Once users have processed jobs, they can subsequently take advantage of other people's computer cycles. Additionally, BOINC's default job scheduling was modified to improve fairness, such as to prefer users processing many jobs.

This system allows an exhaustive definition of jobs while leveraging a cycle-sharing platform into a global computer cycle market.

Keywords

BOINC, volunteer computing, grid computing, Peer-to-Peer, Virtualization

Resumo

Atualmente, beneficiar da partilha de ciclos de computadores através da Internet é difícil para utilizadores comuns, mesmo usando sistemas bem conhecidas como o BOINC. Isto deve-se ao facto de ser difícil para utilizadores comuns instalar a infraestrutura necessária, desenvolver as aplicações de processamento e adquirir suficientes doadores de ciclos. Em geral, os proprietários de computadores só têm um papel no sistema: doar o poder de processamento dos seus computadores.

Nesta tese, descrevemos um conjunto de extensões de BOINC que facultam a qualquer utilizador a criação e o envio de trabalhos que tiram proveito de ciclos remotos. Para submeter os seus trabalhos, os utilizadores apenas necessitam de fornecer os ficheiros de entrada, definir a aplicação de processamento, bem como os argumentos que serão fornecidos à aplicação. Posteriormente, outros utilizadores contactam o servidor, recebem um conjunto de trabalhos, e processam esses trabalhos numa máquina virtual segura. Consequentemente, todos os trabalhos são processados dentro de um ambiente isolado e deste modo não comprometem o computador de processamento. Isto é necessário, uma vez que os utilizadores podem enviar qualquer tipo de aplicação. A máquina virtual e todos os ficheiros necessários para o trabalho são transferidos usando o protocolo BitTorrent. Os ficheiros devem ser hospedados pelo criador do projeto. Da mesma forma, todos os ficheiros de saída gerados são enviados diretamente para o criador do projeto, minimizando assim, os ficheiros salvos no servidor BOINC. Utilizadores que processam trabalhos, posteriormente podem tirar partido dos ciclos de computadores de outras pessoas. Por isso, o agendamento de trabalhos foi modificado para ser mais justo preferindo, por exemplo, os utilizadores que processam muitos trabalhos.

Este sistema permite que uma grande variedade de trabalhos diferentes sejam submetidos, criando a partir de uma plataforma de partilha de ciclos um mercado global de ciclos de computadores.

Palavras Chave

BOINC, computação voluntária, computação em grid, Peer-to-Peer, Virtualização

Contents

1 Introduction **1**

1.1 Motivation . . . 2

1.2 Contributions . . . 7

1.3 Thesis Outline . . . 7

2 State of the Art **9**

2.1 Distributed Computing . . . 10

2.1.1 Volunteer Computing . . . 10

2.1.2 Grid . . . 11

2.1.3 Cloud Computing . . . 14

2.1.4 BOINC . . . 14

2.2 Incentive Mechanisms . . . 14

2.3 Data Distribution . . . 16

2.4 Virtualization . . . 17

3 nuBOINC Architecture **21**

3.1 Boinc Integration . . . 22

3.2 REST API and Web Application . . . 25

3.3 nuBOINC Manager . . . 27

3.4 nuBOINC Application . . . 28

3.5 Scheduler and Feeder . . . 29

4 nuBOINC Implementation **33**

4.1 Technologies . . . 34

4.1.1 nuBOINC App and Manager . . . 34

4.1.2 Javascript Web Application . . . 35

4.1.3 Design . . . 35

4.1.4 Server Software . . . 36

4.1.5 Configuration files . . . 36

4.2 Detailed Implementation . . . 37

4.2.1 nuBOINC Manager . . . 37

4.2.2 nuBOINC Application . . . 38
4.2.3 REST API and Web Application . . . 41
4.3 File size and transmission . . . 44

5 Evaluation **45**
5.1 Usability . . . 46
5.2 Transfer Overhead reduction . . . 48
5.3 Deployment . . . 49
5.4 Experiments . . . 51
5.4.1 Credits . . . 51
5.4.1.A Experiment One: Granting Credit . . . 51
5.4.1.B Experiment Two: Prioritization . . . 52
5.4.1.C Experiment Three: Prioritization with elapsed time . . . 53
5.4.2 Conclusion . . . 53

6 Conclusions and Future Work **55**
6.1 Conclusion . . . 56
6.2 Future Work . . . 56

Bibliography **59**

Appendix A Server Config File **A-1**

Appendix B Job Info File **B-1**

Appendix C Server Changes **C-1**

List of Figures

1.1 From BOINC wiki: How BOINC works . . . 3
1.2 DNA@Home total credit statistics, 2011-2014 . . . 4
1.3 BAM - sign up for project (2014) . . . 5
1.4 Extended BOINC . . . 6

2.1 BOINC Architecture . . . 10
2.2 Full Virtualization . . . 18
2.3 Para Virtualization: additional layer between Guest OS and HyperVisor . . . 18
2.4 From the Docker website: Virtual Machine - Container difference . . . 19

3.1 Extended BOINC Architecture . . . 23
3.2 Website displaying projects from all users . . . 25
3.3 Website displaying user info and list of all users . . . 26
3.4 Website explaining how to use of the REST API . . . 27
3.5 nuBoinc Manger: list of user projects . . . 28
3.6 Inverse Exponential Decay Formula; with maxcredit=200.000 . . . 30
3.7 Example of task selection over a time period . . . 31

4.1 XML vs JSON vs YAML . . . 37
4.2 Exemplification of torrent padding . . . 38
4.3 nuBoinc Manger: dialog to select files and tracker for torrent creation . . . 38
4.4 job execution . . . 39
4.5 detailed virtual machine setup . . . 40
4.6 new database tables and their connection . . . 42
4.7 work submission and generation . . . 43
4.8 package auto-completion . . . 43

5.1 Website account registration . . . 46
5.2 nuBOINC Manager: select files and generate torrent . . . 46
5.3 Job creation: torrent upload . . . 47
5.4 Job creation: show torrent files and executable selection . . . 47
5.5 Job creation: job configuration . . . 48

5.6 Job creation: virtual machine configuration 48
5.7 Job creation: set job memory/disk space requirements and input file names 49

List of Tables

3.1 Example of task selection 31

5.1 Credits after processing jobs from User1 52
5.2 Credits after processing jobs from User2 52
5.3 Created jobs database order and priority 52

Abbreviations

BOINC Berkeley Open Infrastructure for Network Computing

BAM BOINC Account Manager

REST representational state transfer

API application programming interface

UUID universally unique identifier

MVC model view controller

HTML HyperText Markup Language

URL uniform resource locator

JSON JavaScript Object Notation

OS Operating System

VM Virtual Machine

1

Introduction

Contents

1.1 Motivation 2

1.2 Contributions 7

1.3 Thesis Outline 7

1.1 Motivation

When computing complex tasks, one is often confronted with long processing times. To solve this issue, the task must be subdivided into various subtasks and be executed on a large number of processors or different computers. Current solutions to obtain such computing power is to either use Super Computers, Grid Computing, Cloud Computing [1] or Volunteer Computing.

Super Computers are clusters that are made of numerous processors and serve for processing a large number of parallel tasks. Additionally, the processors are in close proximity and thus it is possible for tasks to interchange data without great performance loss.

Grid computing collects the resources of computers from various locations to compute tasks. The grid is a distributed system comprised of numerous separate computers generally provided by institutes and universities. Grids can be confined to a network of workstations inside a corporation or be large and distributed across many companies. When using tasks that depend on each other and interchange data, those confined grids are more effective than the wider systems. The distance between computers in wider grids makes it infeasible for tasks to interchange data in short periods. The most effective way to use wide systems is the use of bag-of-tasks parallelism. Meaning that all tasks are independent, similar and maybe have different input files or arguments.

Cloud Computing is a general term for large groups of remote servers that are interconnected and can be adapted to provide various types of services without any deep knowledge of each underlying technology. It is easy to obtain a machine to process some datasets. However, to setup a system on a larger scale some expertise is required.

In most cases it is required to obtain an authorization to access such systems, pay for them and have some programming knowledge. This makes it infeasible for normal users to process their own data sets.

Another possibility to obtain computing cycles is to use Volunteer Computing: a distributed system in which computers are provided by people who voluntarily donate their computer's processing power to one or more projects. Here only bag-of-tasks parallelism should be used. The computers are distributed across various locations, each with a different Internet connection. Furthermore, it is not assured that the computer is always connected to the Internet. The owner of the computer might even decide to power it off during the execution of tasks. However, the use of their computers is free, since they owners *volunteer* to donate their computer's processing power. This would allow normal users to use a distributed computing system without requiring to obtain an authorization or needing to pay.

Berkeley Open Infrastructure for Network Computing (BOINC) [2] is a platform for the distribution of parallel jobs to be executed on remote computers. Many Institutions are already using the platform to distribute their tasks, which require considerable computing power. Their projects do research in areas such as Astronomy, Physics, Biology, Mathematics and numerous other areas.

The remote computers are provided by users who want to support specific projects, such as Folding@Home [3] which tries to find cures against cancer, Alzheimer's and Parkinson's, by folding proteins and therefore require many calculations by donors. In order to participate in a BOINC project, donors have to download and install the BOINC client on their computer. Afterwards, they can attach their

computer to the project though the client.

Whenever jobs are available, the BOINC client will first verify if enough disk and memory space is available. Afterwards, the steps as shown in Figure 1.1 are executed.

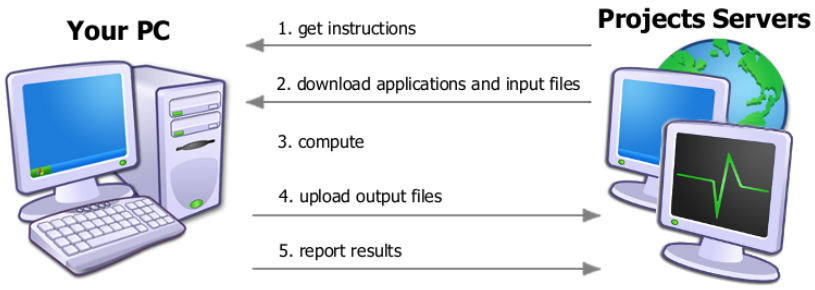


Figure 1.1: From BOINC wiki: How BOINC works

The BOINC client gets instructions from the server, then downloads the applications and input files and executes it as defined in the instructions. Later, the generated output files will be uploaded to the server and the client reports that the tasks are finished.

A list of many known projects using BOINC can be found at the BOINC website [4]. There are also a few private projects listed, but in general, it is difficult for normal users to create a successful BOINC project. In fact, some computer user communities could take advantage of remote idle cycles to speed up their jobs, but do not have the skills to efficiently use BOINC. For instance, users can be non specialists or designers that use ray tracing software to render movies or images, or researchers that use statistical softwares to process very large data sets.

In order to use BOINC, one has to first acquire a server and a DNS Name, since the BOINC Client does not allow raw IPs. To setup the BOINC server, Linux should be used, users have to understand user and group permissions, have knowledge of mySQL and familiarize themselves with the BOINC functionalities to submit applications, update them and configure the application's output and input files.

To efficiently create the data sets, an application needs to be developed that subsequently calls BOINC functions to submit the work into the system.

Finally, when developing the BOINC application, users can choose to use the BOINC wrapper to run their application without using the BOINC API, or choose to develop an application adapted specifically to BOINC.

After these preparations, donors can register themselves to provide their computer's processing power to the project.

The problem emerges when normal users want to create their own project. They need to have a high profile in order to attract enough donors and they also need to know C++ programming. Projects that cannot satisfy these requirements will not be able to take advantage of available remote cycles. Benefits will be low for short-term projects, and for projects not capable of attracting donors. Even if the project is able to initially attract donors, if their interests declines and no new users register themselves, the project starves.

It is possible to observe user and credit statistics on the BOINCstats website [5]. Though, one should

be careful to make assumptions about the project status. For example FightMalaria@Home [6] seems to be dead, but is actually finished as can be inferred from their website news and forum messages. However, their plan is to begin a new research and only rename the project in the hope that current users will support it and not remove the project from the BOINC Manager.

Another project named DNA@Home [7] also seems to be dead, as can be seen in Figure 1.2. The figure shows the total credit generated each month from 2011 to 2014. During that time, no additional credit was generated. But actually it is being merged into another project and therefore no new jobs are directly generated from DNS@Home. In summary, the listed projects are successfully acquiring donors, sending jobs and processing the results.

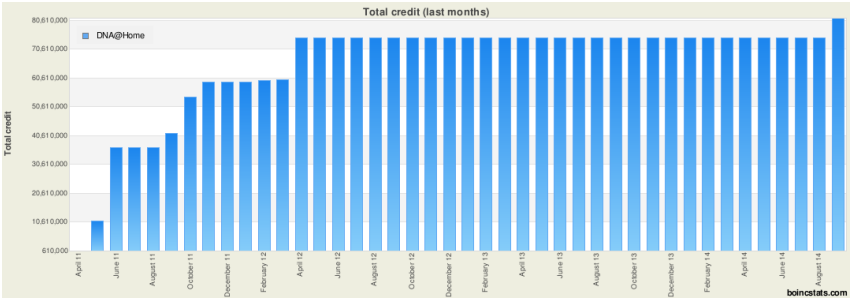


Figure 1.2: DNA@Home total credit statistics, 2011-2014

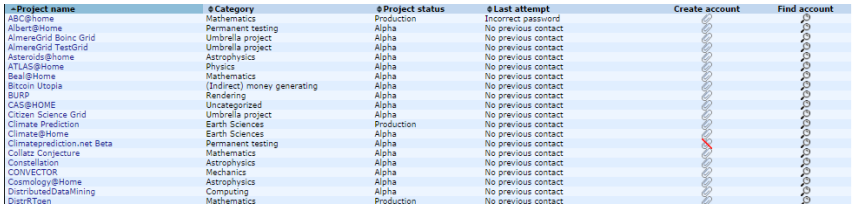
Although this may be true, one can note a similarity between the mentioned projects: they try to maintain the current user base, leave the website online and keep the users informed. This is due to the fact that each BOINC user has to manually find and add projects they want to support. If a project shuts down and starts another similar one, it is highly possible that they loose all current donors and need to start from scratch.

To support projects, donors have to search for them and find the required URL to insert into the BOINC Manager. The BOINC Manager handles registration. Some users might initially be enthusiastic to keep track of projects and search for new projects. At the same time this can be a lot of work and over time the user might become disinterested.

From the perspective of the project creator, the same issues are relevant. They have to keep up the interest of users and assure a constant feed of jobs so that users are sure that the project is ongoing.

Some attempts have been made to circumvent this issue, an example is the BOINC Account Manager (BAM) [8] which simplifies the process of attaching new BOINC projects. Users do not have to search for new projects, instead they can see a list of projects and choose which to support (see Figure 1.3). Then again, users must select the projects manually in order to support them. When selecting a project through BAM, the user can either let it create an account on the project's website or search for an existing one.

Additionally, the only way to submit a new project is by creating a post in the forum. A special thread exists for this purpose, but simply posting there does not mean that the project immediately appears on the BAM project list. At the moment, many projects can be found with a forum entry age between some days and 6 years, some of which are already retired, meaning that no more jobs are created. It is unknown if the projects attained their goal or wether other issues arose, such as not having enough



Project name	Category	Project status	Last attempt	Create account	Find account
ABC@home	Mathematics	Production	Incorrect password		
Albert@Home	Permanent testing	Alpha	No previous contact		
AlmereGrid Boinc Grid	Umbrella project	Alpha	No previous contact		
AlmereGrid TestGrid	Umbrella project	Alpha	No previous contact		
Asteroids@home	Astrophysics	Alpha	No previous contact		
ATLAS@Home	Physics	Alpha	No previous contact		
Beal@Home	Mathematics	Alpha	No previous contact		
Bitcoin Utopia	(Indirect) money generating	Alpha	No previous contact		
BURP	Rendering	Alpha	No previous contact		
CAS@HOME	Uncategorized	Alpha	No previous contact		
Citizen Science Grid	Umbrella project	Alpha	No previous contact		
Climate Prediction	Earth Sciences	Production	No previous contact		
Climate@Home	Earth Sciences	Alpha	No previous contact		
Climateprediction.net Beta	Permanent testing	Alpha	No previous contact		
Collatz Conjecture	Mathematics	Alpha	No previous contact		
Constellation	Astrophysics	Alpha	No previous contact		
CONVECTOR	Mechanics	Alpha	No previous contact		
Cosmology@Home	Astrophysics	Alpha	No previous contact		
DistributedDataMining	Computing	Alpha	No previous contact		
DistrRTgen	Mathematics	Production	No previous contact		

Figure 1.3: BAM - sign up for project (2014)

donors. For the purpose of maintenance, another thread exists where users can report retired projects. Again, this can be a lot of work. Additionally, users searching for new projects, must read the 'new projects' thread periodically in order to then add them in the BOINC Client.

Normally, a BOINC project must develop an application for each Operating System (OS) and adapt the code if it uses system specific features. Moreover, users are unsure if the application will cause problems on their computer. To solve this issues, BOINC already promotes the use of a Virtual Machine (VM) as job execution environment. Any OS can run inside a VM, therefore reducing the code development cost, as users do not have to adapt the code for various different Operating Systems or learn a new programming language (C, or Java).

Furthermore, the execution environment will be entirely separated from the host computer, allowing users to install any required software and at the same time assure the owner that the computer will not be compromised [9–11]. Nonetheless, BOINC projects need to provide a virtual machine that already contains the required software to run the data processing application [12]. The application must be provided separately and will be executed inside the VM.

In order to facilitate the use of a cycle-sharing system, users should be allowed to use the applications or programming languages they know, and there should be enough cycle donors available immediately to speed up their jobs. The project creator should not need to perform any type of announcement or advertising, specially for small projects. Moreover, donors intending to support all kinds of projects should not need to search for them over and over again. Additionally, they should not fear having malicious code executed on their computers. Consequently, all applications should be run in a secure environment, e.g. a virtual machine. Still, project creators should not need to provide a special VM, but rather just define the software requirements for their jobs.

In this thesis we present extensions to BOINC that allow every user to submit jobs into a public generic infrastructure. Users will have two roles: on the one hand as the owner of jobs that are to be executed on remote computers, and as the owner of the computers where jobs will be executed.

In order to accomplish this, a few changes to the BOINC server software were made, and a custom BOINC application, additional server software, a browser client and a manager that handles the result were developed. The project is named nuBOINC.

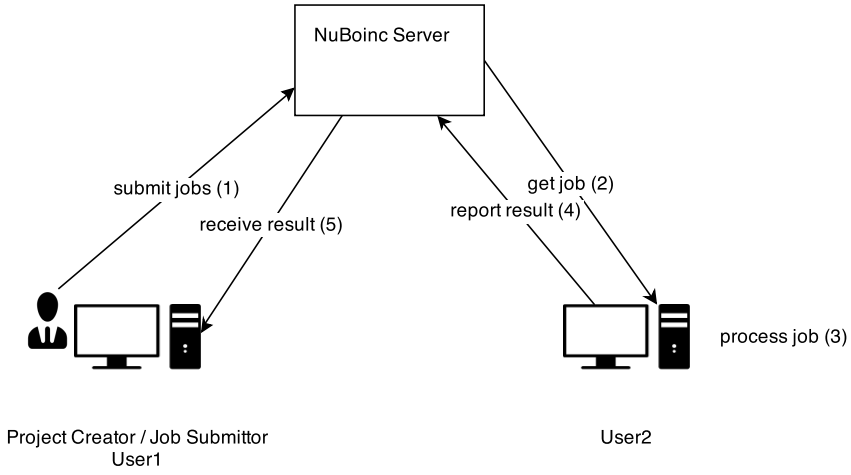


Figure 1.4: Extended BOINC

As shown in Figure 1.4 there are two roles when interacting with our extended BOINC server:

- Users that submit jobs and that receive the results through the nuBOINC Manager (user1): Project creators.
- BOINC users that execute the jobs (user1 and user2): Donors provide their computers' CPU cycles to the nuBOINC server to process jobs. Users can be both donors and project creators.

To submit and create new jobs (1), project creators use the developed website, which allows easy creation of bag-of-tasks, or use the REST API, which gives much more control to advanced users. The server waits for BOINC client requests to distribute each job. When contacted (2), it checks which user is requesting work and selects jobs according to this information.

Clients receiving the job information invoke our application which downloads required files and then executes the job (3). Finally, the results are returned (4) and can be obtained by the project creator (5). After donating cycles in nuBOINC, a computer owner will receive credits and be able to take advantage of remote cycles to speed up his own jobs. This compels users to provide more and more cycles to others. This is further promoted by giving users with a high quantity of credits a higher initial priority when they need the remote cycles (this is capped, however). On the other hand, users with fewer credits will also have low initial priority.

In the event that numerous users with a high priority create many tasks, a system is needed that does not cause a total preemption of low priority tasks. To guarantee fairness, the actual rank will be obtained by using the time difference since the creation of the task plus the initial priority, consequently tasks that were created some time ago can be ranked higher than newly generated tasks. This new relationship within the system will increase the number of users and the amount of time each user is disposed to share with remote users.

From preliminary experiments, we can conclude that these extensions allow the definition and execution of many different kinds of jobs, all of them taking advantage of remote idle cycles. We managed to execute a batch of video rendering, adding effects to the video parts, as well as running simple Python scripts. In the next section, we present existing cycle-sharing platforms, some work on incentive mecha-

nisms and how they relate to our proposed solutions. In later sections, we present the extensions made to BOINC and their evaluation. Finally, we present the conclusions and discuss future enhancements.

1.2 Contributions

The work and results presented in this thesis have made improvements such as:

- A BOINC architecture that allows users to easily submit jobs and to process those jobs.
  - a simple modern website for job submission, user login, user registration and to display info about projects and users. The code adheres to the conventions set by the used framework and therefore can be easily extended.
  - A REST API that adheres to a flat URL convention. The website communicates with it and also allows advanced users to setup their own work generator.
- A modified job scheduling mechanism that imposes fairness by taking into account the users that submitted the tasks and how much work they have processed.
- All required files are transferred by using the BitTorrent protocol. Consequently, no files except the torrent file are saved on the server.
- A VirtualBox virtual machine in Stream mode. This reduces size by 70% and additionally has the effect that all changes will be saved in a separate location. This allows us to downloaded it only once and then provision it separately for each job.

1.3 Thesis Outline

This document is organized in the following way:

- State of the Art: in this section, a review of related research will be presented. It is divided into four subsections: Existing Volunteer and Grid Computing projects that allow users to submit files; Incentive Mechanisms; Data Distribution and Virtualization.
- Architecture: the description of the architecture of nuBOINC. It describes the various newly developed components.
- Implementation: details about considered technologies to develop the various components and their implementation.
- Evaluation: here we evaluate the usability of the developed modules and test the modified scheduling algorithm.
- Conclusion: this chapter presents this thesis' conclusions and describes future directions of development and research.

2

State of the Art

Contents

2.1 Distributed Computing 10

2.2 Incentive Mechanisms 14

2.3 Data Distribution 16

2.4 Virtualization 17

2.1 Distributed Computing

As discussed in the previous chapter, different systems exist to execute and speed up parallel job. In this Chapter will be discussed the approach existing systems take to allow users to submit tasks and how their incentives mechanisms work.

2.1.1 Volunteer Computing

BOINC [2] is the best known platform for the creation and execution of distributed computing projects, providing all the data storage, communication and client management infrastructure. The architecture can bee seen in Figure 2.1.

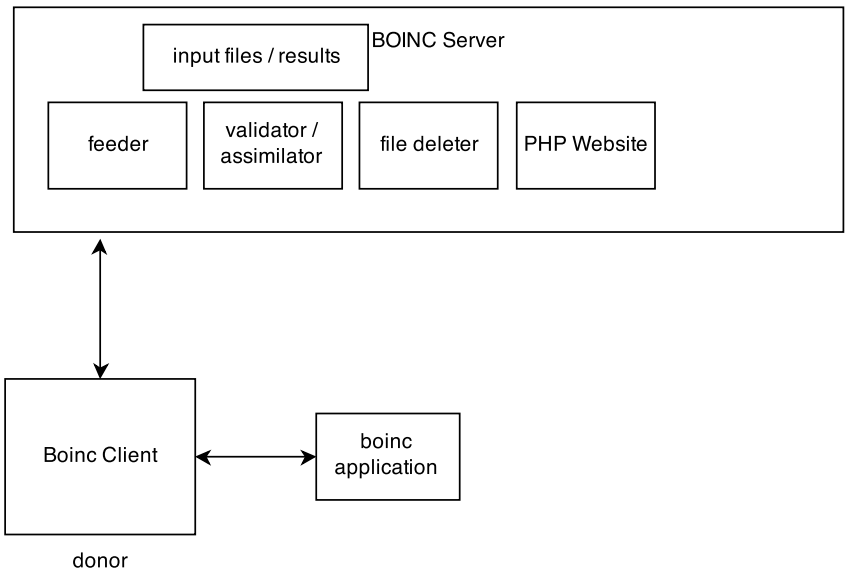


Figure 2.1: BOINC Architecture

The project manager has to program a C++ application (boinc application) that will be downloaded by the BOINC client and then executed on the donor's computers to process data. Later the output files are uploaded to the BOINC server. In addition, a validator [13] and an assimilator [14] should be implemented. These programs are responsible for verifying the correctness of the output files and for processing them. BOINC provides very basic versions of these programs: the validator only checks for the existence of the output files and the assimilator copies results to another directory. In order to create jobs efficiently, an application needs to be developed that calls BOINC specific functions.

Even though it is easy to install the project hosting infrastructure, two issues arise: it is necessary to have C++ programming knowledge to create the applications and, in order to have some speedup gains, it is necessary to publicize the project so as to attract clients.

BOINC wrappers [12] allows the use of legacy applications to process tasks in a BOINC project. Project developers only have to define the configuration file where it says how the legacy application will be executed.

Both the wrapper and the legacy application are downloaded by the client, and when executed, the

wrapper invokes the legacy application. Even with this solution, short-term projects or those without capacity to obtain cycle donors cannot take advantage of BOINC.

BURP [15] and Leiden Classical [16] are BOINC Projects that allow users to submit their own input files to be processed. BURP's aim is to serve people requiring computing power for rendering 3D-animations and images. It provides Blender for rendering. The input file must be a Blender-file containing a database of all the necessary information. These files must be uploaded to the server through the website. To obtain the results, a download interface exists.

Users can write about their projects and results in the forum, some will even be displayed on the main web page. This will also motivate more donors to support the project.

In fact, some open movies are being rendered and re-rendered through this project, such as *Big Buck Bunny* [17] and other movies announced on the BURP website. Through BURP it is possible to render movies in 4K resolution, as well as 3D or other formats that would require a considerable amount of time on one computer.

The BURP project is currently active as can be inferred from the BOINC Stats website. It shows the number of users steadily increasing (currently about 6000 users), however only around 250 users are actively contributing.

Leiden Classical allows to test simulations of various molecules and atoms in a classical mechanics environment. This project was especially made for students at the Leiden institute. A program named Classical-Builder [18] was developed to create input files for the program Classical. Additionally, it allows to directly submit files.

Even so, users are only able to submit input files for predefined processing software. They are unable to submit their own custom software on any of those systems. Furthermore, no incentives are provided that compels users to process jobs. They do not have to obtain credits to pay for the use of the platform nor do they need to obtain priority for the processing of their submitted jobs.

However another form of prioritization exists in both systems: BURP prioritizes jobs such as to finish some last server side computations sooner whereas Leiden Classical creates a queue for each user.

2.1.2 Grid

A great deal of research has gone into systems that allow users to submit their own specific jobs [19], including the submission of custom processing software and data sets. Unfortunately, most are grid systems and are either not applicable to public cycle sharing systems or they are neither currently publicly available nor hosted.

For instance, Condor [20] (HTCondor since 2012) allows the execution of any submitted job on remote computers on a LAN. Users that want to submit jobs must have access to that central entity, either by attaching their own computer to Condor or by having access to a computer with Condor installed. In both cases, an administrator needs to grant access. All computers participating in the Grid have Condor installed, but run different daemons. Finally, users need to create a "Submit Description File" as shown below.

```
Executable   = foo
Universe     = standard
Log          = foo.log
Queue
```

This file must then be submitted with condor_submit. The executable and input-files path can be relative to the current working directory or be an absolute path. The files will be transferred as soon as a condor-worker receives the task.

A more complex example can be made as follows:

```
universe     = vanilla
Executable   = povray.$$(OpSys).$$(Arch)
Log          = povray.log
Output       = povray.out.$(Process)
Error        = povray.err.$(Process)

Requirements = (Arch == "INTEL" && OpSys == "LINUX") || \
(Arch == "X86_64" && OpSys =="LINUX")

Arguments    = +W1024 +H768 +Iimage1.pov
Queue

Arguments    = +W1024 +H768 +Iimage2.pov
Queue

Arguments    = +W1024 +H768 +Iimage3.pov
Queue
```

Here the universe is set to vanilla, such as to not stop the process in the middle of execution. Three jobs are queued, each with different arguments. The requirements specify that it must only run on Linux systems. Similar to BOINC, a central entity manages the submitted jobs and decides where to run them.

The priorities of the jobs are determined by the user's priority. Initially, they begin with the best priority, but this can be individually changed by the administrator. Each users receives a share of all available machines based on their priority. If users use more machines than indicated by their priority, it will worsen over time. Conversely, if they are using less machines, their priority will improve. No relation exists between the user's priority and jobs executed on their machines. In other words: their priority does not increase by allowing Condor to run jobs on their computer.

Condor is mainly focused on High Throughput Computing (HTC), as the new name suggests. Therefore, it is not suitable for volunteer computing which is distributed over large distances. It would reduce the throughput. Moreover, Condor assumes that all connected computers are trusted and are constantly connected. Workers do not queue jobs and the scheduling policies do not involve jobs already being

executed on computers that may disconnect at any time.

Another system is CompuP2P [21], which uses a decentralized system similar to BitTorrent to distribute the tasks. It uses a market oriented mechanism to sell computing resources to users that require them. It is well-suited for heterogeneous environments on the Internet. Users that wish to submit jobs must create an XML file that describes how the tasks should be executed and the maximum price offered for successful execution. After submitting the file, a service layer will parse it and will try to find a seller and forthwith manage all communication between the seller and the buyer.

Like other systems [19] investigated, no public software is available. POPCORN [22] and G2-P2P [23] provide a programming API that a developer must integrate directly into their application in order to use remote cycles. Although POPCORN exists, it is not maintained anymore.

OurGrid [24, 25] is a grid middleware based on a peer-to-peer architecture. It consists of four components that can be installed on every computer: Broker, Peer, Worker, Discovery Service.

A peer is responsible for determining which machines can be used as workers. The worker component runs on the machines that will execute tasks. If a peer does not find a suitable machine in its network, it can use the discovery service to find workers in other networks. The discovery service is already installed on OurGrid servers. Finally, the broker allows to submit and monitor jobs.

Anyone can submit jobs by creating a Job Definition File (jdf) and submitting it into the systems. The jdf must have the following format:

```
job :
label  : SimpleJob2

task :
init   : put Fat.class Fat.class
remote  : java -cp . Fat 3 261147332 6819792792357414911 output-$JOB.$TASK
final   : get output-$JOB.$TASK output-$JOB.$TASK

task :
init   : put Fat.class Fat.class
remote  : java -cp . Fat 3 261147332 6819792792357414911 output-$JOB.$TASK
final   : get output-$JOB.$TASK output-$JOB.$TASK
```

Inside the ”init” clause the command ”put” defines which files are to be transferred to the worker node, the remote clause indicates how to invoke the processing software and the final clause defines with the ”get” command which determines the files to transfer from the worker back to the user’s machine. The jdf can then be submitted with the broker application.

The incentive mechanism is based on favors and credits. Each time a node allocates resources to a requesting consumer it does a favor. Accordingly, the consumer becomes indebted with the owner of the resources. If the dept grows, it will be less prioritized by that node. Each node keeps track of a local balance for each known node. Even so, it is still possible for non-contributing nodes to access the

resources of the system, but only if other nodes with more credit are not requesting them.

Registered users can choose to run jobs in a virtual machine to be on the safe side or non-virtualized to provide the best performance. However, the website status shows that only 20 workers are currently available (9/2014). Which stands no comparison with some other BOINC projects (i.e asteroids@home) that have thousands of active users.

2.1.3 Cloud Computing

Finally, Cloud Computing can be used. It can be described as follows: *"A computing Cloud is a set of network enabled services, providing scalable, QoS guaranteed, normally personalized, inexpensive computing platforms on demand, which could be accessed in a simple and pervasive way."* [26]

To conclude, Cloud Computing is a very general term and thus can be adapted to solve various problems. For example to setup a small private cluster to process datasets. This aspect is already being investigated and provided by various enterprises [26, 27]. For instance, Aneka [27] provides a framework to develop distributed applications. They provide an API for developers to access their system resources.

This is also being provided by Amazon [28]. They call it High performance computing in the cloud and claim that it can be scaled to tens of thousand instances. Additionally, they demonstrate in a video how to build an 8-node cluster to run a molecular dynamics simulation.

In the video, the setup is done in a detailed manner. But Amazon also provides a system to ease the setup by providing templates that describe the required structure of the system.

However, those systems require users to either develop an applications that uses an API or to correctly setup the infrastructure. Under those circumstances, it will be complex for normal users to use such systems.

2.1.4 BOINC

This leaves BOINC as the most successful system, which is due to “the use of two natural human reactions: empathy with the problem being solved, and competitiveness among users”. [19]

With this in mind, we extend BOINC so that users already supporting other projects can easily join nuBOINC.

As moral incentives for supporting a project, such as finding cures, does not exist in this project, the rewarding system must be improved to provide better incentives.

2.2 Incentive Mechanisms

The most critical aspect of an incentive mechanisms is to enforce corporation, since users tend to attain as much resources as possible without contributing to the system. These users are called free riders [29] and are also a known problem in Peer-to-Peer (P2P) systems, which thereupon provide incentives to users.

Commonly two types of incentive mechanism are used in P2P systems: market oriented and reciprocation [29]. Systems that are market oriented use economic models to control resource distribution. Users need to pay in a form of virtual currency to receive services from other peers. CompuP2P [21] is such a system and defines the terms ”seller” and ”buyer” to represent nodes that process tasks and nodes that submit tasks. Some nodes will be assigned to be ”market owners” and do the job of a matchmaker between sellers and buyers. Buyers that have insufficient credit cannot use any resources. The reciprocation approach rewards participants based on their contributions, which is done by OurGrid [24] (see previous section). It uses the behavior of peers to decide if tasks should be assigned to them. This can be based on direct reciprocity or indirect reciprocity. Direct reciprocity bases its decision on services received from a specific node, whereas indirect reciprocity takes into account the services that a specific node has provided to other nodes.

BitTorrent's [30] incentive mechanism limits resources to free riders and gives more resources to users contributing to the system. In detail, this is solved by using a Choking Algorithm [31] which is based on the tit-for-tat policy [1]. When a peer 'X' connects to another peer 'Y' it will initially registered as a choked peer. Any peer has a limited amount of upload slots that are dynamically assigned to peers requesting chunks. When peer 'Y' decides to assign a slot to a peer it will unchoke the peer with the highest upload speed.

Even so, this might be unfair for users with low upload capacity. For this reason an effort based algorithm was suggested that includes the capacity of each peer to decide if it should be unchocked [32]. If their upload speed is high in comparison to their capacity, they will be ranked higher. An analogous method can be used in volunteer resource sharing systems [33] where CPU time slots are interchanged. Here the algorithm also includes the total CPU slots to decide to which computer it should send a task.

In BOINC, users only provide resources. The CPU time is not divided into individual slots that can be dynamically allocated to jobs. BOINC assumes that each job will use one process core. Nonetheless, BOINC provides credits as incentive as well as inducing mechanisms to fairly distribute credits. This is done by using the actual FLOPs performed by the application. It tries to be device-neutral, so that jobs running on different systems or CPU/GPU receive the same credit, additionally host should get the same credit per day regardless to which project it is attached to [34]. Hence, jobs that run longer time on specific systems will still receive the same amount of credit [35] as jobs that run a short time on more potent systems.

However, to retrieve the correct FLOPs value, a BOINC specific API function must be called. It is not always possible to call this function, i.e. when using the BOINC Wrapper, or to know the actual FLOPs performed. To workaround this issue, a benchmark is first performed on the host, with which it is possible to approximate the FLOPs performed by an application.

Then again, P2P differs from nuBOINC in some aspects: In general, P2P is decentralized and the time period in which the incentive algorithms can be applied is limited (until the download completes), whereas in nuBOINC a central server is present and the time period for an incentive algorithm can be infinite (the projects' total duration). This could possibly limit the applications of P2P incentive algorithms.

[1] tit-for-tat is an English saying meaning ”equivalent retaliation”

Another problem in incentive system is the Sybil attack [29] that occurs when providing some sort of reputation to users and they can take advantage of it. Users could create many identities to gain more influence over the system. For instance, they can create new identities when the maximum reputation is achieved, especially when the reputation is based on currency that can be consumed. In such a case the total currency can be further increased by creating more identities. Another case, called white washing, can happen if new users start with a reputation that may decrease. In this case, users create new identities if their reputation drops below the initial one.

Problems such as free-riding and Sybil attacks on P2P systems are common and some solutions have been proposed [36, 37]. The conclusion is that the current mechanisms are not sufficient and a trustworthy/more reliable entity is needed. This entity is called a Super Node [36, 37] and manages a currency and reputation system. It has also been mentioned that using currency has the side-effect that users would be able to accumulate currency for future use. Furthermore, it is just as important to prevent users from being able to achieve and maintain a dominant position. Even users without malicious intent, contributing to the system will raise their amount of currency indefinitely and consequently dominate the system. Therefore, the currency and reputation need to be limited to guarantee balance.

A similar mechanism can be applied to nuBOINC. Here, free riders are users who only submit jobs, but are reluctant to process any from the system. The Super Node is the central server and the credits represent the currency to compute the priority or pay for tasks to be executed—even though this is not as dynamic as in a P2P system, since only one server is available. To prevent Sybil attacks and free riding in nuBOINC, users must begin with zero credits and receive some limitations when submitting jobs. This will be discussed in Section 3.5.

2.3 Data Distribution

Projects that have large datasets and are using BOINC need a huge amount of disk space to save and process the results. Moreover, a good Internet connection is desirable in conjunction with a stable server. When a BOINC project gains more popularity, the user base increases and could exceed the servers capabilities. BOINC already proposes some improvements to increase the server's capacity [38]. The main advice is to use a multiple host system to distribute the load. Another research [39] analyzes the performance of a BOINC project with small datasets and also proposes to optimize database queries. However, their analysis is limited to datasets of a size below one gigabyte. For projects that have larger datasets, the network bandwidth might become a bottleneck [39].

One proposed solution is to use a Content Delivery Network (CDN) [40] to mirror the files from the BOINC server. Henceforth, file requests to the BOINC server would be redirected to a surrogate server near to the client. Output files from clients would also be uploaded to certain data centers. Of course, this setup would cost more than the normal single server setup. Some projects already use replicated and distributed servers to solve this issue [39].

Another proposal is the use of BitTorrent. In this case, each input file is represented by a corresponding torrent file. This file will be downloaded by an altered BOINC client that includes torrent capabilities

and thus can download the required file. By using this method, clients could effectively increase their help to the project by sharing downloaded project files and consequently reduce the server load [41, 42].

However, this only applies to distributing files. The results must be uploaded individually to the server. Another key point is the limitation of peers that are allowed to host and propagate data. This is done in order to reduce the risk of malicious users sending bad data. On the other hand, BitTorrent already verifies the content authenticity and if it is not valid the data will be downloaded again. Nevertheless, bad data would still reduce the system's performance. Another problem is that a client-to-client connection is needed, which are often blocked by NAT or a firewall.

In relation to nuBOINC, this solutions can be adapted. Especially to distribute files, BitTorrent can be used to easily let project creators host files on their computer without being required to pay a server. However, a proposed solution is to limit the peers that are allowed to host and propagate data. This limitations cannot be applied to nuBOINC, since every user who wants to create a project must also be allowed to host data. Project creators could also create their own "CDN" by seeding the input files from various servers. Likewise, in nuBOINC the output files will be sent to a specific server, for example to the computer of a project creator.

Although this may be true, some general issues still remain. Those are the firewall/router configuration needed for torrent to work and malicious users trying to send bad data through torrent.

As mentioned, the BitTorrent protocol already prevents bad data download by checking the SHA1 hash of the pieces received. [30]. One problem is that torrents referring large files will create large file pieces and increase the amount of bad data a user can send. A workaround is to force the torrent generator to create small pieces. However, this will create torrent files with increasing file size, since numerous hashes need to be saved in the file. This can be avoided by using merkle torrent [43], which constructs a hash tree of the content. By doing so, the size of each piece will be 2log of the file-set size. Another issue is the Choking Algorithm used. Using it is not desirable in nuBOINC. All donors should be able to download the files in equal manner, without being impaired by not having enough upload speed or not uploading at all. Therefore, the project creator should disable the choking algorithm on his side.

One aspect of the router configuration issue is that some routers can be configured remotely by using UPnP [44]. This includes opening and closing ports and it is used in nuBOINC.

2.4 Virtualization

A virtual machine is a software implementation of a physical computer. It supports the execution of a complete operating system in an isolated space. They are used to test operating systems or unknown applications in a safe way. They can also be used as a sandbox to run any application in a safe environment, such that it cannot access the main system. There is a great deal of different software solutions to enables the creation of virtual machines, e.g. VirtualBox [45], VMware [46], KVM[47] and XEN[48].

VirtualBox and VMware only provide full virtualization. XEN and KVM offer both full virtualization and para virtualization. On full virtualization systems, the hardware is completely emulated within the

guest operating system whereas in para virtualization (PV), a software interface is presented to the virtual machine. This interface allows the guest to run specific operations on the hardware without going through virtualization. Moreover, PV does not require virtualization extensions from the host CPU. Instead, the guest system must be modified in order to work with the provided interface, adding a layer between the Guest OS and the hypervisor as seen in Figure 2.3 compared to full virtualization in Figure 2.2.

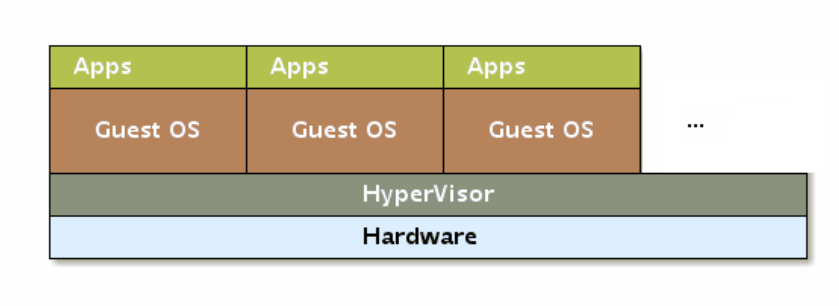


Figure 2.2: Full Virtualization

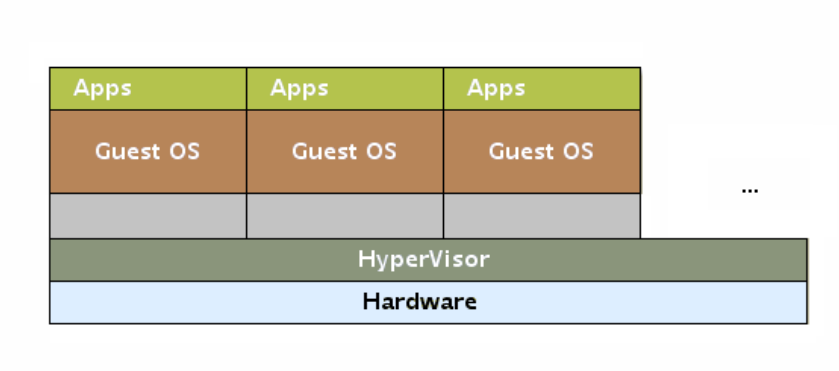


Figure 2.3: Para Virtualization: additional layer between Guest OS and HyperVisor

Another possibility is the usage of Linux Containers [49], used by Docker [50] to separate user space on Linux servers, including limitation of memory and disk space. The advantage of the Linux Containers (see Figure 2.4) is that no actual virtualization takes place, but at the same time the software environment inside a container is completely isolated. This also makes it ideal to sandbox applications without any performance loss. Moreover, many systems resources can be used and do not need to be replicated in each container, consequently saving space compared to virtual machines. However, the containers only run on Linux and can only run Linux applications. Moreover, no state can be saved.

We will use the properties of virtual machines to run any application in a safe way as well as making it easy for developers, as they only need to target one operating system. In case various hypervisors need to be supported, LibVirt [51] could be used. To setup a virtual machine, however, an extensive configuration file needs to be created.

In this thesis, we propose the usage of VirtualBox. Not only is it already included in the BOINC installer, it also provides an easy API and some important features. First of all, VirtualBox runs on all the well known operating systems like Windows, Mac and Linux, it also supports hardware virtualization

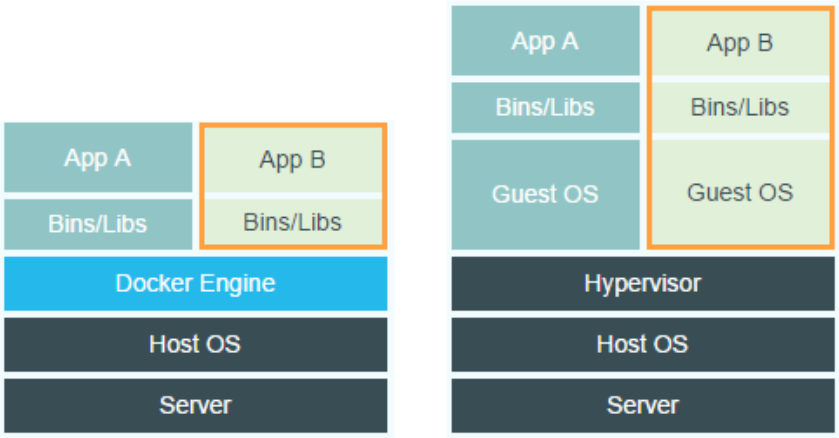


Figure 2.4: From the Docker website: Virtual Machine - Container difference

(Intel's VT-x and AMD's AMD-V) which theoretically reduces the Guest OS [2] performance costs in terms of processing cycles. Additionally, VirtualBox provides the functionality to save and resume the machine state.

Furthermore, it allows marking a medium as immutable [52], consequently saving all changes to a different location. This feature in particular allows the creation of multiple machine instances of machines with the same disk, but with different processes running on each instance. Another feature is the use of shared directories. This allows the sharing of directories between the guest and the host, enabling the guest to write to a specific directory on the host. Ultimately, these features are used to save space and time when executing nuBOINC tasks.

As for security in VirtualBox, a large amount of work has been done investigating how malicious codes can affect the host and how to prevent this [53–55]. Currently, most of the security issues are related to security of the host and the hypervisor. Another thread is the detection of the virtual machine emulator and executing specialized code directed to it. The network is also a target of malicious code. Finally, one should also mention that the VirtualBox Guest Additions are targets of malicious code and there were reports of vulnerabilities [56]. In nuBOINC the network is only active during the process of package installation, afterwards it is turned off and therefore prevents executed code to access the network.

The shared folders in itself will not give direct access to the Host system, but an application could create many big files and thus use up all the disk space, whereas a virtual machine limits the disk space usage. The BOINC client already verifies if a job does exceed a certain bound of disk space. If it does, the job will be aborted [57].

Another key point is that a user should not try to open unknown files in such directories, outside the virtual machine, since they could contain malicious software.

[2] OS running inside VirtualBox

3

nuBOINC Architecture

Contents

3.1 Boinc Integration 22

3.2 REST API and Web Application 25

3.3 nuBOINC Manager 27

3.4 nuBOINC Application 28

3.5 Scheduler and Feeder 29

The BOINC architecture allows institutes with a high profile to benefit from Volunteer Computing. However, it is difficult for normal users to setup a BOINC server and, especially, to acquire donors for they project.

Therefore, BOINC should be extended such as to create a public infrastructure that could be setup only *once* and would benefit all persons that require computing cycles.

The BOINC architecture should allow anyone to create projects and submit processing applications, input files and define jobs. This is currently not possible with the BOINC web interface. Consequently, the interface needs to be extended with those functionalities. The BOINC wrapper [12] could then be used to process the applications in a secure environment.

However, each job would require to create different BOINC applications, job templates [58] and job input files. Not only would this increase the complexity of the server, but also the amount of files saved on the server.

Therefore, by taking advantage of BOINC functionality allowing to specify an URL in the job template for each input file, the files should be hosted by the project creator. Then again, it would require the project creator to obtain a server and pay for it. It would be possible to use an own computer to host the files, but it is not assured that the IP address of the computer remains the same throughout the whole process (receive jobs - send output files). Normal ISP clients receive IPs in a dynamic fashion. It is possible to acquire a fixed IP, but it costs additional money. Additionally, the network bandwidth from normal users generally have low upload capacity, which would reduce the performance of the system.

For this reasons, BitTorrent [30] should be used. It is made for such dynamic environments and distributes the network load. On the other hand, the BOINC client does not include torrent capabilities.

Given these points, a custom BOINC application needs to be developed that downloads files using BitTorrent, executes the job in a secure environment and finally send the output files to the project creator. No additional files and applications are required to be stored on the server.

3.1 Boinc Integration

To integrate user job submission a new and modern website was required such as to reduce and ease the steps to create and submit jobs, and to improve user experience. Additionally, it should be possible for institutes or advanced user to setup their own work generator. This two factors aid the idea of a REST API.

In nuBOINC, only external users would submit and receive files, consequently it is not required to save files on the BOINC server. Nonetheless, a system is needed that can reference input files and download them. The same must be applied to the output files generated by tasks. They will not be saved on the BOINC server and thus must be send to a location specified by the project creator. Therefore, the project creator would be in charge to setup a server to host input files and receive output files. Both functionalities are provided by the nuBOINC Manager.

It was decided that no change should be made to the original BOINC client, such as to not bother users to install another client and thus brush them off. The objective of nuBOINC is to take advantage

of the existing user base that can effortlessly add nuBOINC to the BOINC client.

For this reason, a BOINC application was developed that downloads all required files and executes the task.

The original BOINC infrastructure architecture can be further divided into various parts as shown in Figure 3.1. The newly developed components are: server application, javascript web application, nuBOINC Manager and nuBOINC application. They were developed in order to allow the use of the original BOINC-Client, so that users do not need to install a new one. The server code has only a few lines changed, which are provided by a patch (see Appendix C). More details will be provided in Section 3.5.

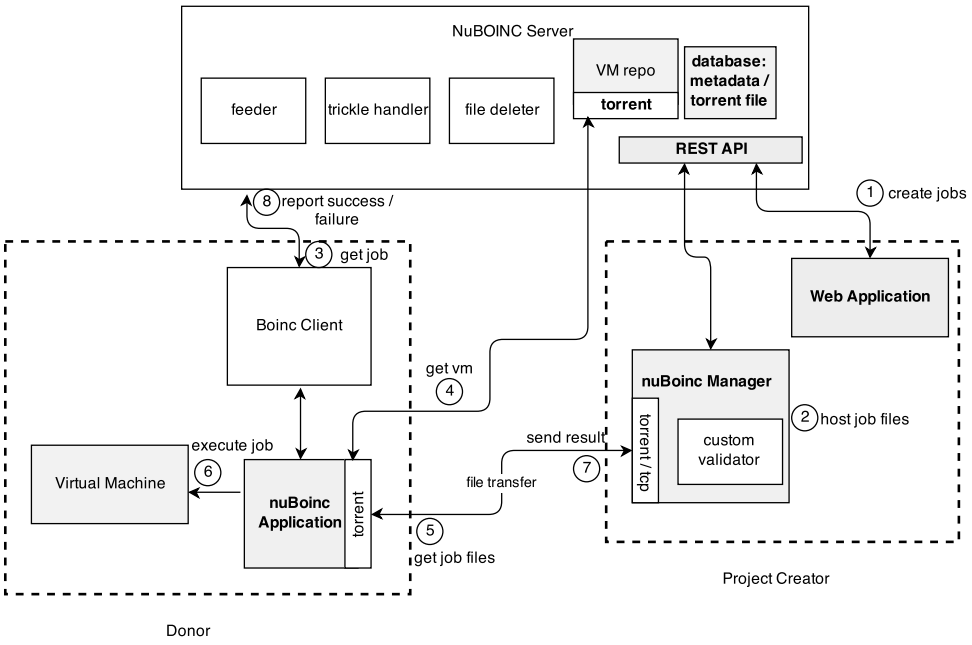


Figure 3.1: Extended BOINC Architecture

In order to allow the execution of user submitted jobs, a BOINC application and an additional server application were developed. The developed BOINC application will be referred as nuBOINC application. Additionally a trickle handler [59] must be setup on the server to grant credits, since the nuBOINC application will run until the results are sent. The default BOINC validator grants credits based on total runtime and no functionality is provided to report the correct CPU time. Even when using undocumented functions in the nuBOINC application to directly report the CPU time, it had no effect. By using the trickle handler it is possible to grant credits only for the processing time of the virtual machine. This is done by sending trickle messages from the nuBOINC application to the server. The messages are then processed by the trickle handler, which can be adapted to process various messages. In nuBOINC, the messages contain the CPU time and the trickle handler grants credits based on that time.

In order to run applications on 64bit and 32bit systems, the nuBOINC application has been divided into two separate applications. A BOINC application consists of various sub-applications, each made specific for an operating system and architecture. For instance a project may have the following applications:

- windows 32bits
- windows 64bits
- linux

Normally a BOINC application can run all jobs on all supported systems. In the case of nuBOINC this must be handled differently.

As an illustration, assume that only one nuBOINC application will be provided which supports both 32bit and 64bit systems. A project creator will then submit a job and choose a 64bit virtual machine to run it, which can only run on a 64bit host. However, a donor with a 32bit system can receive the job, since it is supported by the nuBOINC application. But the job will fail to run, because the 64bit virtual machine cannot run. Some 32bit hosts could support a 64bit virtual machine, but it depends on the hardware and BOINC does not verify this. Moreover, virtual machines that are 64bit and are executed on a 32bit system are slower [60]. Additionally, to access the VirtualBox API, the correct architecture must be used.

Consequently, two nuBOINC applications exists: one BOINC application consists of only one version for 64bit systems and another BOINC application consists of two versions for 64-and 32bit systems. When a user selects a 64bit virtual machine the 64bit BOINC application will be chosen so that the job only runs on 64bit systems. Otherwise, the 64&32bit version will be selected since the 32bit virtual machine can run on both 32 and 64bit systems without problems.

The nuBOINC application as well as the nuBOINC manager communicate with the newly provided server through a REST-based API. They interchange IP addresses so that the output files can be send to the project creator. The PHP files are still accessible since the default BOINC client communicates with them.

The original BOINC web interface was extended with a javascript web application, which allows any user to create their own custom projects and submit jobs. Additionally, it includes the basic user registration and login functionality.

A user that requires remote computing, can use the website to create a project and submit jobs (1). Afterwards all required files need to be made available to donors by using the nuBOINC Manager (2).

When a donor requests a job, it will receive information regarding the job files, processing software, arguments and virtual machine (VM) configuration (3). The BOINC client then downloads the specified VM, if not already present, (4) and afterwards the required job files (5). All are transferred using the BitTorrent protocol. The virtual machine will then be started, all requirement will be installed and subsequently the job will be executed (6) . Later, the output files will be send directly to the project creator (7). Finally, the job finishes and the BOINC client reports a successful execution (8).

Some modifications were made with respect to the job information organization within the server. In a normal BOINC setup, work is grouped into projects and all jobs from the same project are executed by the same application. With this extension, all user submitted jobs are processed within the same BOINC project and by two applications, but they belong to different user projects. For each job there is one workunit (torrent file and execution parameters) and several replicas of each workunit, called results.

To support this organization the database was extended. In particular, only new tables where created and no original table structure was modified, more details regarding this will be given in Chapter 4.

With this extension, the workunit input files contain the job description and a file that referrers to the input files for the nuBOINC application, which then selects and downloads the input files and executes user submitted job. Another difference is that no files are saved on the server. The input files are hosted by the project creator and the output files are also sent directly to the project creator.

It is also important to note that BOINC uses mechanisms to prevent credit falsification [11, 13]. In other words: methods to mislead BOINC in assuming that much more work was carried out. The mechanisms mostly use the BOINC validator which provides different methods to verify the result. But in nuBOINC no real result is sent to the server (only a dummy file) and consecutively the validator cannot perform any real validation . In nuBOINC, the simplest version is used that only verifies the existence of the result (the dummy file). As a matter of fact, even if files were to be saved on the server, it would be almost impossible to use a validator since it must be adapted to every user project.

3.2 REST API and Web Application

The server provides a REST API, which is consumed by the Javascript Web Application, the nuBOINC Manager and the nuBOINC application. The main purpose of the API is to create nuBOINC projects and Jobs. In particular, the REST API call to create jobs is a wrapper for the BOINC work generator script and thus can be used to create any type of Job.

Additionally, the REST API allows the intercommunication between the nuBOINC Manager and the nuBOINC application.

Since the web interface now allows more control over workunits, the security has been increased by providing temporary API-Tokens on login. This token is generated by using UUID4 python implementation, which uses the system's random generator, has an expiry time of 10 minutes and needs to be sent on every request. Two different types of tokens exists to differentiate between normal users and the nuBOINC Manager.

The new developed website has all required functionalities to submit jobs. Additionally, it lists all projects, jobs, users and their credits and the design was chosen so as to give a modern look and feel

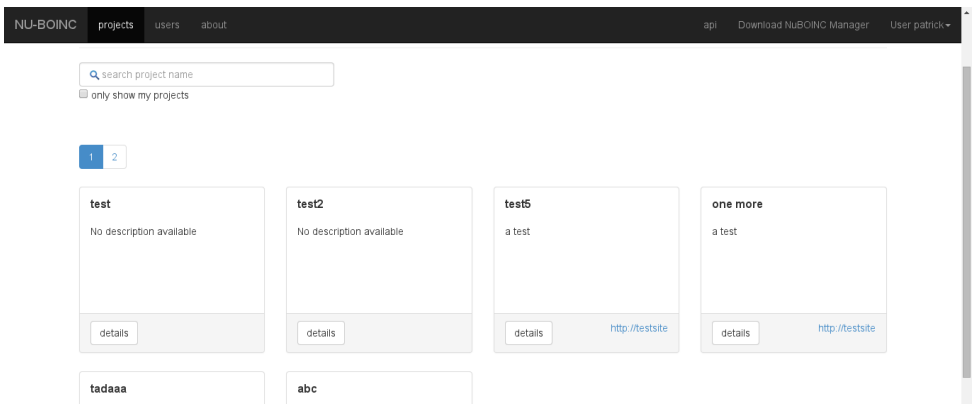


Figure 3.2: Website displaying projects from all users

In Figures 3.2, 3.3 and 3.4 we give an overview of how the website is displayed.

As shown in Figure 3.2 users are able to see all projects, their description and a possible link to a website representing the project. This site supports paging, meaning that the project list is only downloaded partially therefore reducing bandwidth, transmission and processing time. This is done by requesting a specific page with a specific quantity of items on the REST API. It is also possible to search for projects by name. This is useful for users with many projects and users that want see which project their are supporting. The project name and user name can bee seen in the BOINC client during the execution of a job.

The details section presents the job configurations or the job creation page, depending if jobs where already created. This section should only be displayed to the project creator. Consequently, it is only available for users that are logged in. If not authenticated users trigger the detail button, they will be redirected to the login page, afterwards they will be redirected again to the details project page. An authenticated user that is not the creator of a specific project will only see the project description.

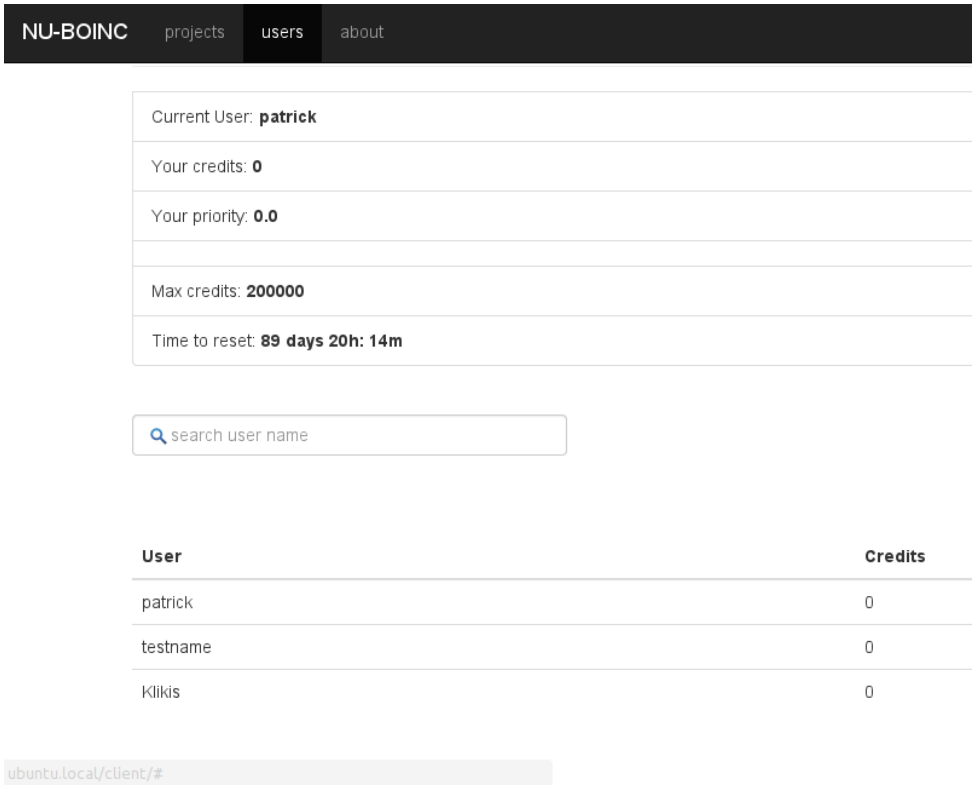


Figure 3.3: Website displaying user info and list of all users

The site displayed in Figure 3.3 shows information about the current logged in user. Here, users are able to track their progress on credits and priority as well as the time until the next credit reset. Additionally, they can see a list of other users to which they can compare themselves. The feature set is similar to the project list page, it supports paging and searching. The search can be used to see the current credits of other known users, for example friends.

The API webpage shown in Figure 3.4 displays information about the REST API. This can be used by advanced users or institutes to setup their own work generator. When triggering the *details* button

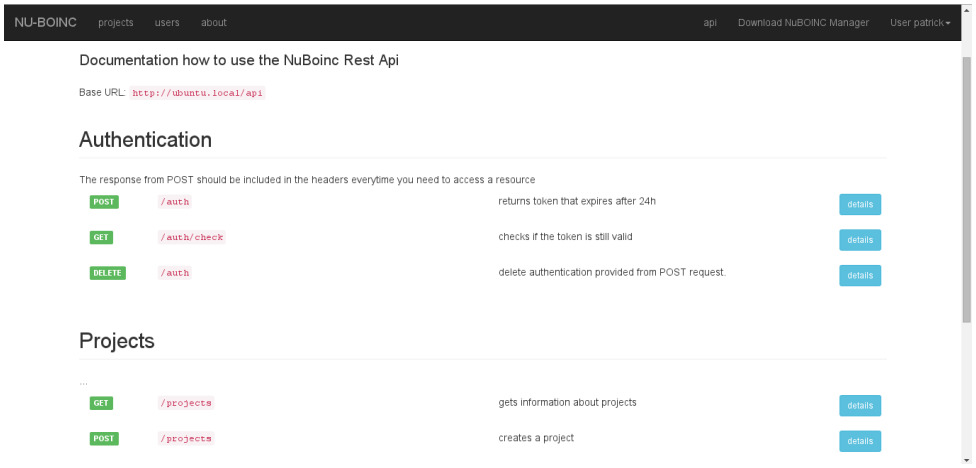


Figure 3.4: Website explaining how to use of the REST API

detailed information is displayed that demonstrates what content is expected in the request and what content will be returned in the response. Currently the REST API only supports JSON. At the moment no request limits are set. Users could generate thousands of jobs.

3.3 nuBOINC Manager

The main purpose of the nuBOINC Manager is to host the input files and receive the output files generated by the various remote clients. This is done by seeding a torrent file and by setting up a local server.

When the Manager is started, it will ask for the user's authentication, login into our server and receive a special token that is valid for one week and has limited access to the REST API. It can only *GET* Projects, Jobs informations and access the communication API. The manager will retrieve the information about the user's projects and jobs and display them as can be seen in Figure 3.5. The *user_project* info property contains the description and website of the project. In the meantime, the manager creates a local server on a specific port, informs the nuBOINC server about it and tries to forward the port on the local router trough UPnP [44], if the service is available.

This is required for client-to-client communication if the computers are behind a router/NAT. Otherwise, it will be impossible for the router to determine to which computer an external request should be sent. When a local computer requests a website, the router automatically assigns a port to the requesting computer and forwards all responses to that port to it. Afterwards the forwarding will be stopped.

When the server receives the port sent by the nuBOINC Manager it can also retrieve the IP number. This information is kept on the server for three hours in memory. No database entry is created. The manager resends the port every 60 minutes, in case the connection was lost and the IP number changed. This data is sent to the nuBOINC application when it needs to send the results.

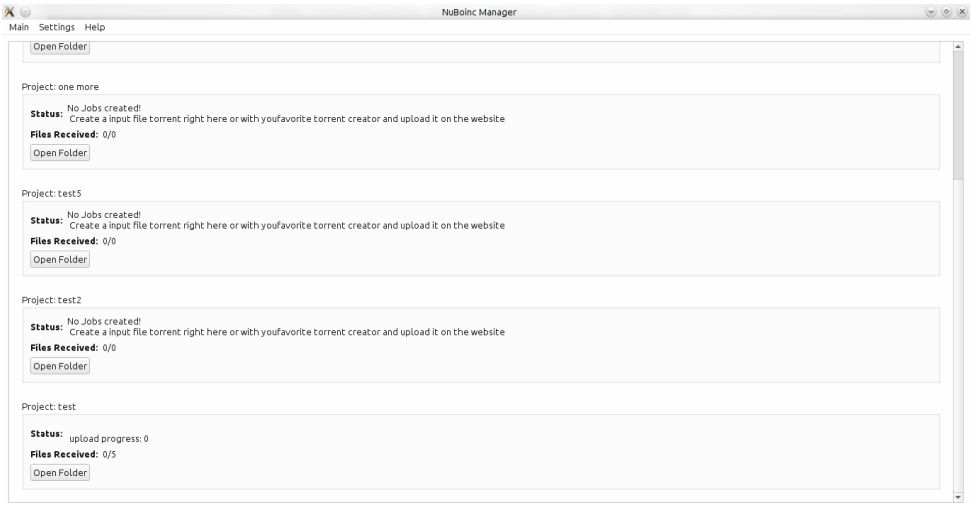

NuBoinc Manager
Main Settings Help
Open Folder
Project: one more
Status: No Jobs created!
Create a input file torrent right here or with youfavorite torrent creator and upload it on the website
Files Received: 0/0
Open Folder
Project: test5
Status: No Jobs created!
Create a input file torrent right here or with youfavorite torrent creator and upload it on the website
Files Received: 0/0
Open Folder
Project: test2
Status: No Jobs created!
Create a input file torrent right here or with youfavorite torrent creator and upload it on the website
Files Received: 0/0
Open Folder
Project: test
Status: upload progress: 0
Files Received: 0/5
Open Folder

Figure 3.5: nuBoinc Manger: list of user projects

3.4 nuBOINC Application

The nuBOINC application is responsible for downloading all required files to execute a job. This is done using the BitTorrent protocol. The file that referrers to the input files is a torrent file that refers all input files and executables for all tasks. The Job description file specifies which files are required for the current job.

Since project creators can submit any type of program, we need to prevent malicious actions. Those include influencing the computer where the script is being executed and any action through the network. BOINC already investigates various security issues that could be exploited by hackers [9, 10].

For one thing a hacker could break into the server and replace the programs with malware. Or the project in itself is a rogue or spoof project and distributes malware programs. The last issues can be solved by using virtual machines whereas server side hack problems must be limited by using code signing. [9, 10]

Therefore, all jobs are run inside a virtual machine, which totally separates the scripts environment from the actual computer (see Section 2.4). After installing all necessary applications, the network to and from the virtual machine will be isolated. This also allows to run the job on any Operating System that supports virtual machines.

After executing a job, our application will try to obtain the IP address of the project creator's computer, which is saved on the server. If no address is available or the transmission fails, our application will retry every 10 minutes. In order to prevent someone from obtaining the address and sending malicious files, an authentication system was put in place.

The nuBOINC application will send the values of *authenticator*, *hostid*, *userid*, *result_name* and *wu_name*. These values are provided by APP_INIT_DATA [61] and if the combination is not correct, the server will not respond. The only person who could still send wrong or malicious files is the computer owner.

3.5 Scheduler and Feeder

To guarantee fairness, the way scheduling is performed has been modified. Users that process large numbers of jobs should receive a higher status when submitting jobs. Nonetheless, users with a low status should not be totally subjugated. To clarify, if they submit jobs, those should be processed at some instance of time.

Additionally, users that submit jobs and have a potent computer connected to nuBOINC conceivably desire they jobs to be processed on their computer as well. Therefore, their jobs should be preferred when requested from their computer. However, they should not receive a higher status when processing they own jobs, since this could lead users to only processing their own jobs.

When contacted by a BOINC-Client the adapted server prioritizes jobs from users who are requesting workunits. If no user jobs are found, the workunit with the highest priority is returned. The priority is calculated using the time difference from when it was created and the user's total credit. The weight on the time difference can be set with the feeder parameter *priority_order_plus_create_time X*, where X stands for the number of days until a user with rank zero should reach a rank of 100. The code modifications can bee seen in Appendix C

In the following example the parameter X is assumed to be 10. The function used is based on the inverse exponential decay formula as shown in equation 3.2. The result is always between 0 and 100. The factor 5 was selected to get the value 100 when a user's credit equals the maximum credit. Each day increments the priority by 10, so that after at least 10 days the priority of the workunit is at 100. The priority may increase further if the job is not processed at that time.

$$\tau = maximumcredit \tag{3.1}$$

$$priority = round(100 * (1 - e^{-user\_total\_credit/\tau * 5})) \tag{3.2}$$

Some possibilities to determine the maximum credit have been analyzed. The first idea was to dynamically determine the maximum credit by selecting the highest credit found of all users. However, this allows a user to obtain a dominant position and handicaps other users when the maximum credit is too high, since they will not see much change in their priority after processing jobs.

Another possibility is to use credits as currency to pay for the processing of submitted jobs. However, it is difficult to determine how much a user should pay for each submitted job. Ideally, the price should be dependent on how much time a tasks needs in order to be processed. In reality, it is impossible to determine this beforehand. Leaving only one option: to pay after the task has been processed. To determine how much credits a task is worth, BOINC runs a benchmark that determines the "Measured floating point speed" [62]. Based on the benchmark one unit of credit represent 1/200 day of CPU time on a computer that does 1,000 MFLOPS. It assumes that a task uses the full processor capacity and therefore only needs the total processing time to determine how much credits should be awarded.

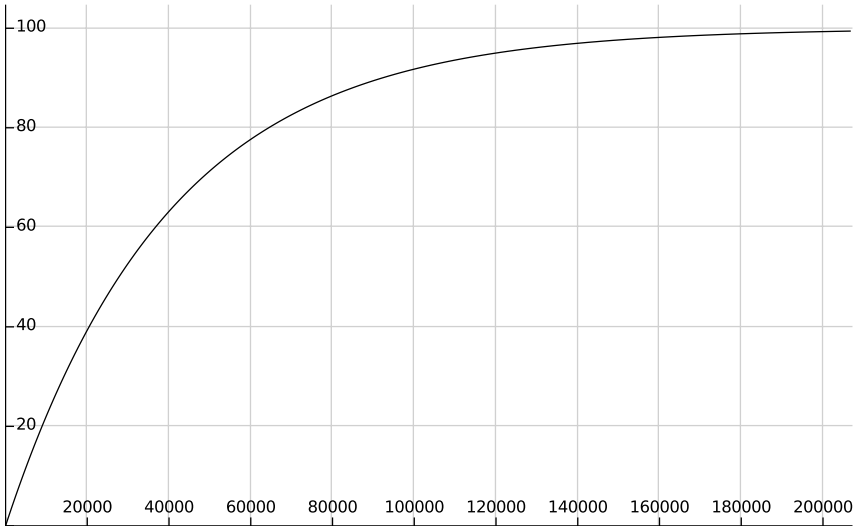


Figure 3.6: Inverse Exponential Decay Formula; with maxcredit=200.000

Finally, we decided to set the maximum credit to a fixed constant, which will be determined by the nuBOINC hoster. This will prevent accumulation of credits and obtaining a dominant position. After a time period, the credits will be reset to zero.

The value of maximum credits should reflect the average credits obtained in a certain time period. Assuming that the current average credit per day is 2000 and a time period of about three months, the maximum credit could be set to 200000 $(2000 \times 100(\ 3months))$. This function (See Figure 3.6) translates into the following:

- All users with a total credit close to the maximum will begin with a high priority but less competition between each other as changes to the total credit will not have a great effect on the priority.
- all users with low credits will have high competition between each other and receive great increase in priority for a raise in their credits.
- a new submitted job from a user with a high priority may have a lower priority than a job from a low priority user that was created some time ago.
- the Sybil attack is useless since users always start with zero credits.
- free-riders do not receive credits and are penalized with possible long wait times.
- prevents accumulation over longer time periods and dominant positioning.

Assume the example given in Table 3.1. Three PCs exists in the network, each belonging to a respective user. In *example 1* all submit 100 jobs but only PC *A* is processing tasks. Since *A* is processing tasks and there are jobs submitted by *A* those will be selected, ignoring priorities. After all tasks from *A* are processed, *A* will begin to execute tasks from others. As shown in *example 2*, the priorities will now be considered by the scheduler. *B* has a higher priority than *C* and therefore his jobs will be selected.

	Example 1			Example 2		
PCs	A	B	C	A	B	C
#jobs	100	100	100	0	100	100
Priority	15	50	30	/	50	30
Workers		A			A	
Prioritize		A			B	

Table 3.1: Example of task selection

Figure 3.7 shows the behavior of the scheduler over a time period. On day 1 the task *B1* was selected and executed. Later *B* creates a new task which receives the priority 30. On the second day task *C1* is selected by having a higher priority than *B2*. Later *C* creates a new task which receives the priority 70. In this instant, the task *B2* was created 5 days ago. It had an initial priority of 30 which increases each day by a total amount of 10. Consequently the actual priority is now 80, which is higher than the rank of *C2* and thus is selected for processing.

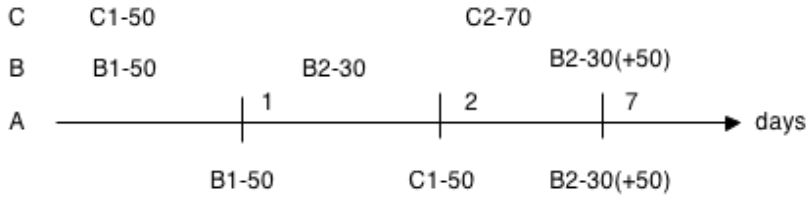


Figure 3.7: Example of task selection over a time period

4

nuBOINC Implementation

Contents

4.1 Technologies 34

4.2 Detailed Implementation 37

4.3 File size and transmission 44

4.1 Technologies

First of all, will be discussed some technologies that have been considered to create virtual machines, javascript web application, server software and libraries to generate and handle torrent files. Later, some more detailed implementation description will be given.

4.1.1 nuBOINC App and Manager

For the nuBOINC application and Manager a few requirements need to be fulfilled as described in Section 3.3 and 3.4). In particular the requirements are that it:

- must be compatible with most modern operating systems, i.e. it must be a cross-platform application
- must have BitTorrent capabilities to create and seed the torrent files
- must be able to communicate with the REST API. The content is always JSON.
- must be able to accept direct TCP connections.

The nuBOINC application has similar requirements, it:

- must be compatible with most modern operating systems, i.e. it must be a cross-platform application.
- must have torrent capabilities to download files referenced by a torrent file.
- must be able to communicate with the REST API. The content is always JSON.
- must be able to communicate with VirtualBox.
- must be able to create direct TCP connections.
- must be able to create a lock file.

First of all, to be cross-platform, the framework QT [63] and wxWidgets [64] have been considered. Both have an extensive API to fulfill most requirements, needed to implement the applications. Additionally, Qt provides file locking functionality and an easy Signal-Slot mechanism to delegate events

However, in wxWidgets the events must be defined in a complex way and does not have functionalities to create a file lock. Both lack a SSH API. Additionally, the VirtualBox C++ API is a bit complex. For Python, on the other hand, exists the module pyvbox [65], which provides a very easy to use wrapper around the virtualbox API. Still, the usage of Qt would greatly decrease development time.

Consequently, the python modules PyQt [66] and PySide [67] were evaluated. Both are python-bindings for Qt. The most important difference is that PyQt supports the Qt version 5.1 which includes the required QLockFile Class. PySide only supports the 4.8 version.

Considering torrent functionality, two libraries have been inspected. The original BitTorrent made by Brahm Cohen [68] or libtorrent [69]. The original BitTorrent is implemented in pure Python, whereas

libtorrent is implemented in C++, but provides python-bindings. A quick view brought out that the code from Brahm Cohen is ten years old, whereas libtorrent is being actively maintained. Moreover it has an extensive documentation.

Another possibility, was the usage of Torrent Sync [70]. But it would require to be installed separately by users and therefore was excluded.

Finally, the decision fell on the combination Python, pyvbox, PyQt and libtorrent. The python script must be directly executable on windows and linux, without user interactions. Therefore, the pyinstaller [71] script is used, which generates an executable that can run without any further requirements or dependencies, since they are all included in the executable.

4.1.2 Javascript Web Application

Javascript Web Applications are more interactive than normal web pages. They can generate and change the content without requesting a new site from the server. If new content needs to be provided, only data is requested from the server. The HTML content is then generated on the client instead of the server and therefore reduces it's load. It is also possible to show different pages when a link is clicked and the URL changes without reloading the site from the server.

Emberjs [72], Angularjs [73] and Backbonejs [74] are frameworks that simplifies the creation of such websites. The use of any framework depends on the user and their philosophies. From various posts and comparisons on the Internet we derived that Backbone is the easiest to learn and is very flexible. On the contrary, Emberjs and Angularjs have a steep learning curve and are less flexible. However they provide a greater feature set, which can be advantageous later on.

In particular, Emberjs is very focused on conventions. It can infer much of the configuration itself and thus reduces possible boilerplate code. Moreover it will be easy for users to contribute to a project using this framework, since they only have to learn the conventions set by Emberjs and not adapt for each different project. This characteristic was the basis for our decision to use the framework Emberjs.

Another aspect of such framework is the separation of the code into various javascript files, but then "compile" them into only one file that will be used on the server. Emberjs also sets some standards for the project directories layout, filenames and template names. When the project needs to be compiled, a task runner such as grunt [75] can be used.

4.1.3 Design

To further improve user experience the design must be modern and intuitive. Bootrap, Ink, Foundation and Zimit are CSS frameworks that improve the design of a website. All provide a similar feature set, but a different design. Nevertheless the design should not be the basis for a decision, since other themes can be applied to the CSS, consequently changing the design. As said, they are frameworks for CSS, not specifically for design. Additionally, all provide responsive CSS, meaning the layout changes for each screen size, e.g. phones, tables or desktops. The final decision fell on Bootstrap. It is currently the only framework which also has direct and extensive Emberjs support [76].

4.1.4 Server Software

Finally, a server that manages many connections at the same time should be used. This is required since each job on each system will make request to the REST API. Additionally, a micro framework would be preferable, since only a REST API will be provided. Furthermore it should support Object Relational Mapping (ORM) to ease the management of the database.

The most used language for server side programming PHP is generally slow. Some project exists to address this issue by creating c-extensions. One project is ”Phalcon” [77], which has a rich feature set and an ORM layer. Another micro framework is YAF [78], which claims to be the fastest framework and could be used together with an ORM framework such as Propel [79].

Another popular language is Python, which is also generally slow, but can now be run with PyPy [80]. A Just In Time (jit) compiler. This drastically improves performance for repetitive tasks. Some micro-frameworks are Bottle [81] and Flask [82]. A popular python ORM framework is SQLAlchemy [83]. This frameworks can also be run with Gevent [84] to increase the concurrency by accepting many connections at once.

We chose to use Python. First of all, in enforces some coding conventions and the ORM libray SQLAlchemy is easy to use and allows to automatically populate the python ORM models with the columns of the database. This is done by using deferred reflection.

This is especially useful since we use an existing database. We only need to define the required models and all properties of the table will be reflected on them. The only models that need to be fully populated are new ones provided for nuBOINC. Furthermore, SQLAlchemy allows to create all specified modules in the database, but without changing existing ones. To improve coding on IDEs that support ”autocomplete”, the attributes of models can even be partially set and later SQLAlchemy will reflect the rest of the properties.

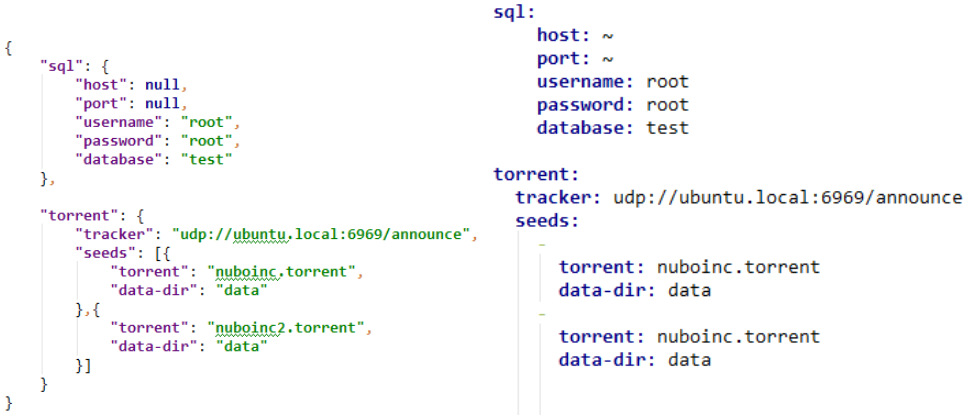
4.1.5 Configuration files

Configuration files must often be edited manually. Usually this can be done with graphical interfaces. However, if the files are located on a remote computer this is not possible and makes it difficult to manage configuration files such as XML. XML is a markup language created for encoding documents. Configuration files on the other hand generally only represent simple data.

Therefore, these configuration files should be written in a data serialization format that focus on being human readable. The JSON [85] markup language uses a human readable text to represent data objects. But indentation and newlines are optional and if neglected the JSON content can again be made difficult to read.

Another data serialization format is YAML [86], which main focus is to be human readable. Indentation and new lines are required and enforce a readable structure. In the Figures 4.1 can be seen a comparison between XML, JSON and YAML.

Conclusively, it was decided to use YAML for the configuration files of the nuBOINC server.



(a) JSON example **(b)** YAML example

(c) XML example

Figure 4.1: XML vs JSON vs YAML

4.2 Detailed Implementation

4.2.1 nuBOINC Manager

The nuBOINC Manager also features torrent creation through libtorrent [69] with a special attribute active: It adds pad files between the torrent files.

This allows any torrent client to individually select files without downloading parts of other files. The reason is explained in [87] and visualized in Figure 4.2: “clients do not transfer files by files but by pieces which have the same size.

At the creation time of a torrent, all the files in the torrent are virtually concatenated and then the resulting data chunk is split into evenly sized pieces, of a size appointed by the torrent making application and indicated in the torrent file. If there is more than one file in the torrent, there might be some pieces containing the end of file A and at the same time the beginning of file B.”

Therefore, if A is selected B will also be selected and allocated on the file system and therefore use up more space. Additionally the nuBOINC Manager creates a torrent that always has a directory, even when only one file is added. The normal behavior is to create a torrent with only that file and no directory.

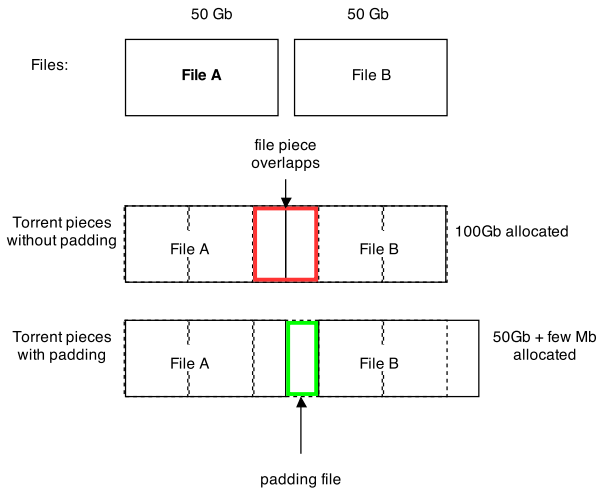


Figure 4.2: Exemplification of torrent padding

The reason for this behavior is explained in (section 3.4). The nuBOINC Manager provides an easy to use interface to generate a torrent from files (see Figure 4.3).

When the manager receives the files, they are saved in a directory with the same name as the job's name and an additional file that contains information about the job. More specifically, it lists the input files, arguments and output files, which are obtained by querying the REST API for the *project_job* information. The response's *info* property contains this informations in JSON format. With this it is also easy to scan the files, identify replicated jobs, compare results and process valid files.

We provide a default "validator" that copies all files from different jobs in one project to another location. This "validator" demonstrates how the job information can be used.

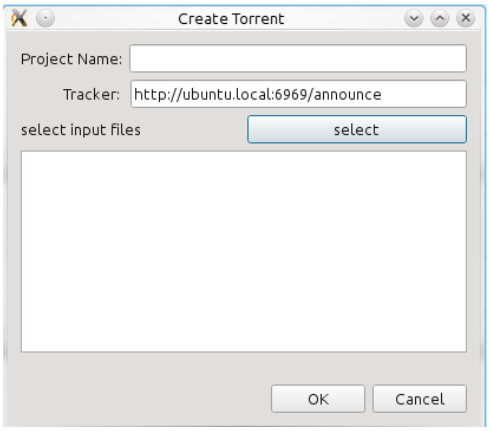


Figure 4.3: nuBoinc Manger: dialog to select files and tracker for torrent creation

4.2.2 nuBOINC Application

When a BOINC client receives a job from the nuBOINC server, it will receive the nuBOINC application, job information as json (Appendix B) and the torrent file submitted by the project creator.

The client invokes our application, which will first verify if an Internet connection is available by pinging

https://www.google.co. If the ping is not successful the function *boinc need network()* will be called and afterwards the function *boinc network poll()* will be repeatedly called until it returns true.

Next, various subtasks will be executed by the nuBOINC application. The Figure 4.4 gives an overview of the tasks. Each step will be further described in the following enumeration.

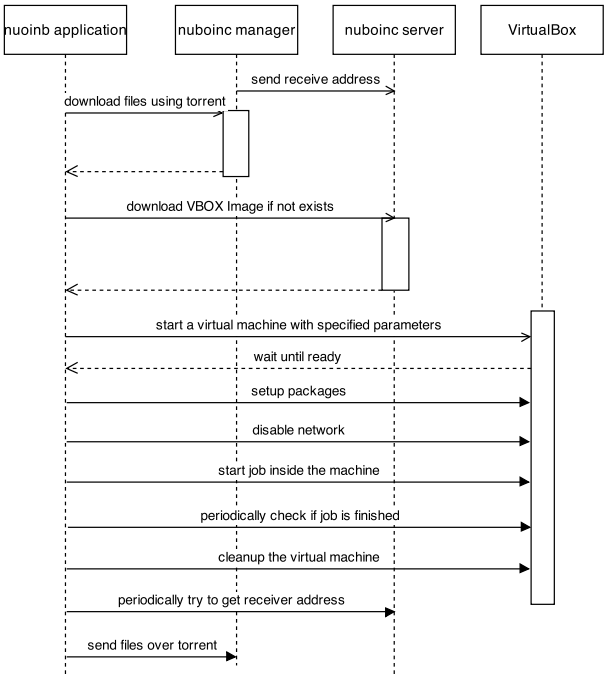


Figure 4.4: job execution

1. Download: downloads the files specified in jobinfo-inputs. The jobinfo-input B contains two informations for the files; *remotename*, which specifies the file in the torrent to be downloaded, and *localname*, the file name as needed by the processing application. The internal torrent client allows to specify which files should be downloaded and which should be ignored. If the torrent was created with the nuBOINC Manager (see section 3.3) no file pieces will overlap and only the specified files will be allocated on disk and be downloaded. If the *remotename* is different from *localname* it will be renamed using a feature provided by libtorrent [69] coupled with the possibility of still seeding the renamed file with the name it had before.

2. Get VBox Image: downloads the specified torrent that points to a VirtualBox image or a zip that contains an image. The image is downloaded to a temporary location which, on Linux, will be cleaned after a restart. Only one process downloads the VirtualBox image. To prevent multiple downloads a lock file is used, which is also saved in the same temporary location. The medium will then be opened in read only mode and set to immutable.

3. Execute (see Figure 4.5): first a virtual machine will be started, gets assigned a random name in the format *nuboinc-randomuuid* and the opened medium will be attached. A network interface of type *Host Only* will also be attached to provide Internet access and at the same time limit the virtual machine's network to the host. The memory is limited to the size specified by the project

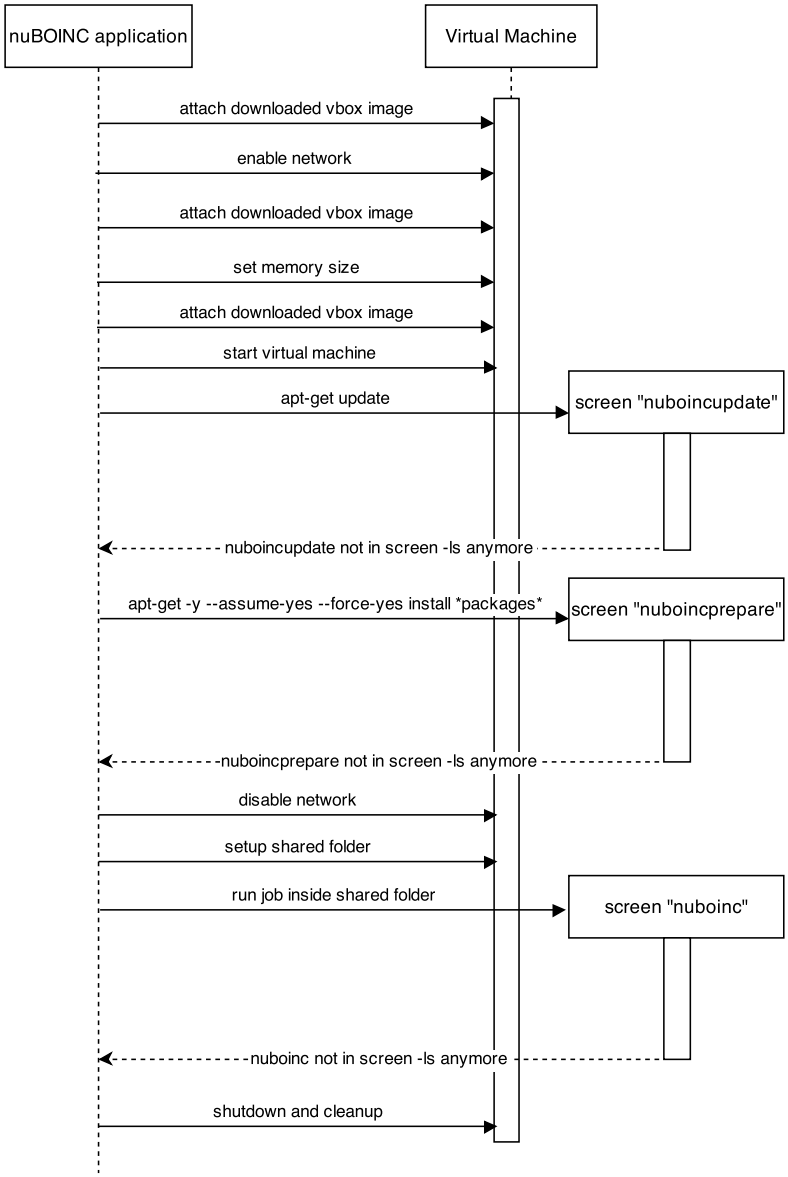


Figure 4.5: detailed virtual machine setup

creator plus 100 megabytes, which are needed by the virtual machine to function correctly. The machine will then be launched in headless mode, so that it does not create a new window. To wait until the machine is ready, it waits for the special property */VirtualBox/GuestInfo/Net/0/Status* to bet set. It logins with the provided credentials in the jobinfo and create a shared folder that points to the directory created by the torrent download. Consequentially the torrent must always specify a directory, even with just one file. To prepare the virtual machine for the processing application we first add any needed repositories with *add-apt-repository* and then execute *apt-get update* in a named screen [88] and periodically test if the screen is still running by calling *screen -ls*. Then any specified packages are installed by executing *apt-get -y –assume-yes –force-yes install *packages** with the same method. Installing package files supplied by the project creator is also supported by calling *dpkg -i *file** After this process the network adapter will be deactivated on the virtual machine. Finally the processing application is invoked with it's arguments.

4. Monitor: The processing application was also executed in a named screen. This subtask periodically polls if it is finished.

5. Cleanup: This process stops the virtual machine and deletes all associated files. The original virtual machine image remains.

6. Send Result: The first step is to obtain the upload link from our server. If it fails, or a disconnect happens during the sending process, it will be retried every 10 minutes. Before sending the file, some informations will be sent first. Those are: The project name, the job id, the filename, the file size, the file's MD5 sum and the result name. One megabyte is reserved for this information. Since this information normally does not occupy the full size, the rest is filled with spaces. When every file is successfully sent, the application exits.

4.2.3 REST API and Web Application

The new server and JavaScript application were implemented with an eye on overall performance improvements, user experience and server load reduction.

As more users begin to use nuBOINC server load increases. A server using PHP with Apache will create a thread for each request, PHP is then invoked in those threads. This causes a big overhead if many connections are being made and the time to process a request is short. Our REST API only transfers small amounts of data at a time and has low processing times.

Therefore our server is implemented in Python using the Gevent library [84], which provides high-concurrency and therefore decreases latency if many connections are made to the server. Gevent does not use threads that are managed by the OS, but Greenlets, which are pseudothreads that are cooperatively scheduled. This means that only one Greenlet is running at any given time. The concurrency is obtained by switching context when reading and writing from or to file like objects. Those include sockets from accepted connections, connection to the database and files. During the writing-and reading process another task can be performed.

It is important to realize that no complex and time consuming algorithms should take place, yet even then it is possible to manually switch the context. This provides high-concurrency and therefore decreases latency if many connections are made to the server, since accepting new connections does not create a big overhead.

Gevent provides monkey-patching to replace the default socket and file functionalities with the ones provided by Gevent. Afterwards, everything is ready to use Gevent. No code must be adjusted. Although this may be true, it is better to change the SQL transaction type to *SERIALIZABLE* due to concurrent database accesses. Provided that multiple *write* operations on the same item are being executed at roughly the same time. If the transaction type is not serializable the write operations will overwrite each other. Another key point is the usage of the pymysql driver [89] instead of the default mysql driver, since it is implemented in C and therefore the gevent patches cannot be applied.

As mentioned in Chapter 3, some new database tables have been created to support the new architecture. The new tables can be seen in Figure 4.6. They can be created by invoking a simple python

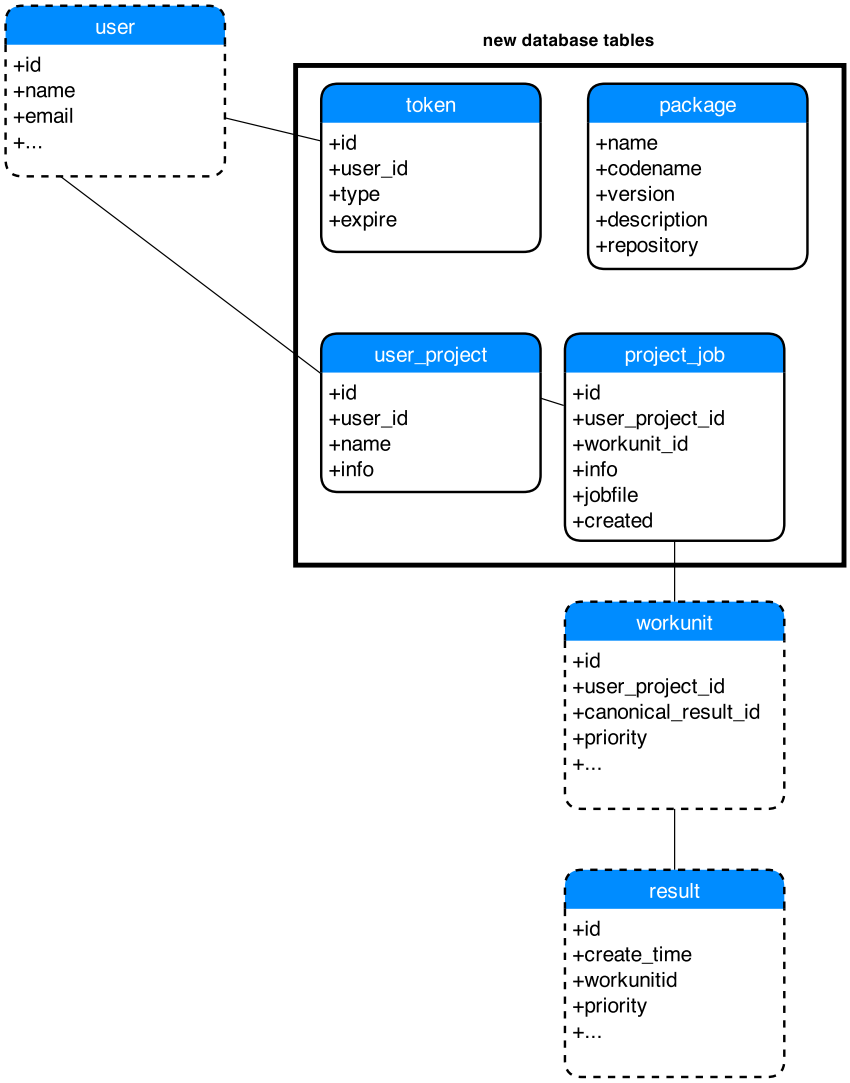


Figure 4.6: new database tables and their connection

script, which uses SQLAlchemy to create the tables.

The REST API allows the creation of projects and jobs as well as obtaining information about them. To create jobs a torrent must be uploaded which can be done either directly with a PUT request or a Form POST request. Also, the uploaded torrents must contain a folder, otherwise they will not be accepted by the server. This is required for the job execution and is explained in Section 3.4.

When receiving request to create new jobs, they are not immediately created (see Figure 4.7), as it would reduce the concurrency, since we would need to wait for the process that creates the jobs to finish. Instead, the jobs information are saved in the database as an *project_job* entry with the *created* property set to *false*. Later, they are created by the work generator process that periodically queries the database for new project jobs.

After calling the BOINC work creator with the supplied informations the *created* property will be set to *true*. The name of the *workunit* contains the name of the project and of the user. This name will be displayed in the BOINC client. As a result, users are able to tell which project they are supporting and from which user and consequentially can look them up on the website.

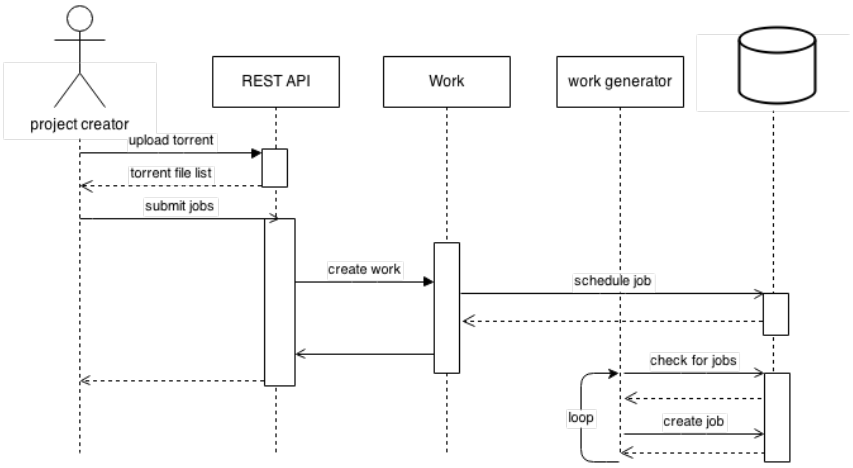


Figure 4.7: work submission and generation

The website was developed with Ember.js [72], a MVC javascript framework with which it is possible to develop a website using HTML templates to represent data. Furthermore, it offers a data persistence library that manages model data and adheres to a set of RESTful URL and JSON conventions. Additionally, to improve the website's design, we used twitter bootstrap [90], a HTML, CSS and JS framework that further improves user experience.

Figure 4.8: package auto-completion

When selecting the packages (see Figure 4.8), the input will show a dropdown list showing packages that begin with the given name. This is done on the server by downloading the package list from `http://packages.ubuntu.com`. The package list is packed in a zip file and is decompressed in memory and then the database will be populated with the entries. This only happens upon the first request, when the database table is empty, or when the last-modified header represents an earlier time than the last update time.

The use of a REST API, offloading the HTML generation to the client and the fact that no files from jobs are saved remotely, allows the use of small sized and inexpensive servers.

It is important to realize that the website is never reloaded for any of the mentioned actions. The data is obtained from the REST API and then the website is updated with the data by rendering the template with it.

4.3 File size and transmission

The VirtualBox image and the input files are downloaded using the BitTorrent protocol [30]. The image is kept in a temporary location and can be used by all jobs at the same time by marking it as immutable. This instructs VirtualBox to keep the changes in a different location and allows our application to download the file only once. The image is provided by our server whereas the job files must be supplied by the project creator. He needs to generate and seed the torrent, which can be done with the nuBOINC Manager.

All input files are listed in the torrent file as defined by the BitTorrent protocol [30] and our application downloads the required ones. During the job's execution, the torrent will continue to be seeded to take full advantage of the BitTorrent protocol. Concerning VirtualBox, some possibilities have been inspected to reduce the size of the disk.

First, a normal Ubuntu Server edition was installed, which used up about 1.4 Gigabyte, secondly a Ubuntu Mini edition was installed, but after installing some required packages it only reduced disk space usage to about one gigabyte. Thirdly, some VirtualBox types have been analyzed. The following exist: *Standard*, *Fixed*, *Split2G*, *Stream*, *ESX*. Even though they are not described in the Manual, we determined that by using the variant type "Stream" for the VirtualBox disk image the size was reduced to around 400 Megabytes from 1.4 Gigabytes (Ubuntu Server), hence reducing transmission -and disk space. The usage of type "Stream" has the consequence that the disk is automatically marked as immutable, which is also exactly what we want. We chose the Ubuntu Server edition since many required packages are already installed.

5

Evaluation

Contents

5.1 Usability 46

5.2 Transfer Overhead reduction 48

5.3 Deployment 49

5.4 Experiments 51

5.1 Usability

The intention of nuBOINC was to create an easy to use interface with which users can submit tasks. Here we list what users need to know beforehand and the steps taken to submit a job.

In summary, users need to know how to invoke their application on Ubuntu since it is the only OS that is supported by nuBOINC. They should be able to test the needed application on the OS and determine which packages and repositories are required. They do not need to specify how to install the packages, this is done automatically by nuBOINC.

When submitting a task, the following steps have to be taken: (as an illustration, screenshots have been taken for most steps.)

1. create an account on the website (Figure 5.1)

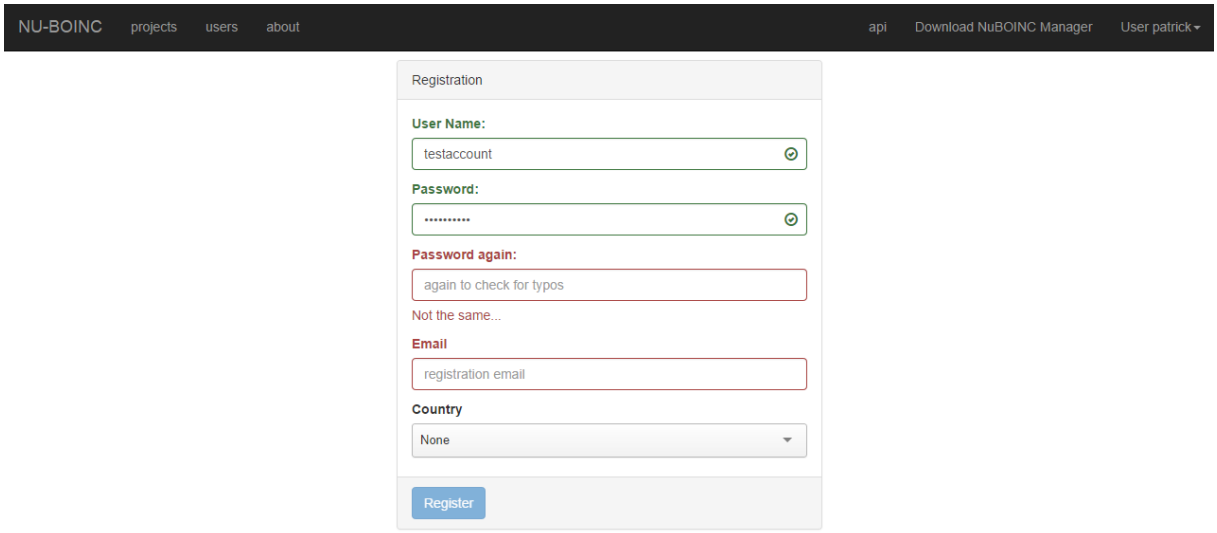


Figure 5.1: Website account registration

2. download the nuBOINC Manager
3. generate a torrent file using the nuBOINC Manager including all input files and the executable, if required (see Figure 5.2)

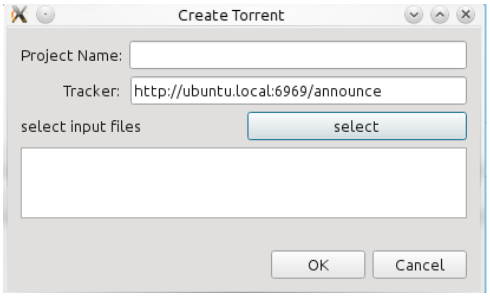


Figure 5.2: nuBOINC Manager: select files and generate torrent

4. use the website to upload the torrent and specify the software to be used and/or the executable provided (Figures 5.3,5.4)

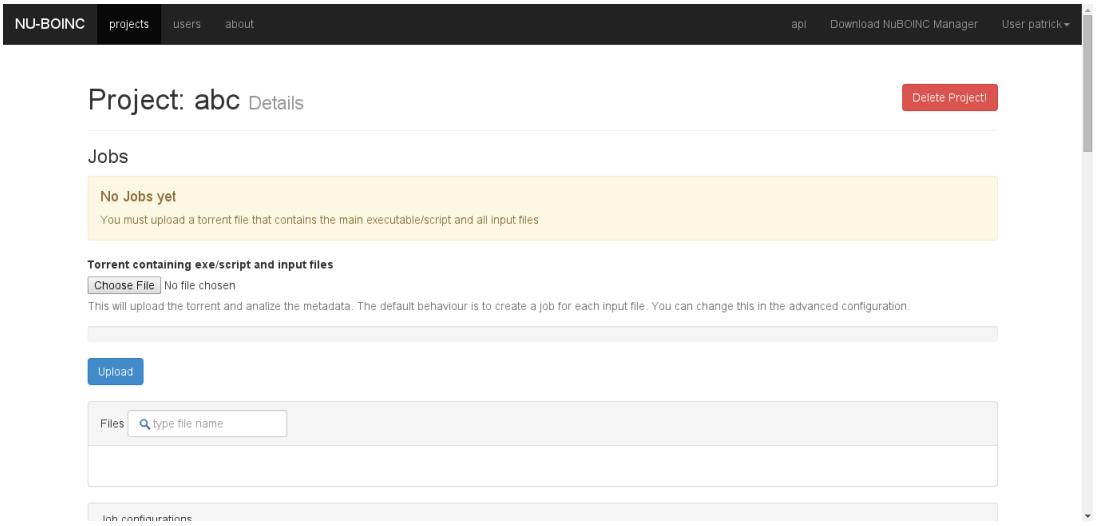

NU-BOINC
projects
users
about
api
Download NuBOINC Manager
User patrick
Project: abc Details
Delete Project!
Jobs
No Jobs yet
You must upload a torrent file that contains the main executable/script and all input files
Torrent containing exe/script and input files
Choose File
No file chosen
This will upload the torrent and analize the metadata. The default behaviour is to create a job for each input file. You can change this in the advanced configuration.
Upload
Files
type file name

Figure 5.3: Job creation: torrent upload

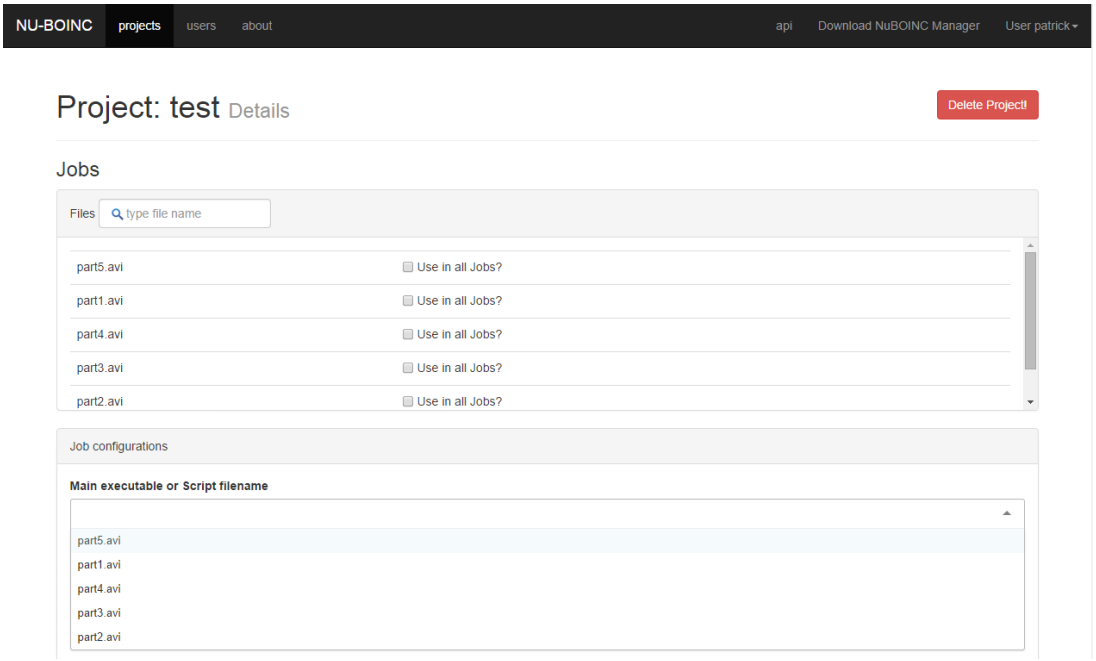


Figure 5.4: Job creation: show torrent files and executable selection

5. define the arguments (if applicable) for the application and the names of the output files (Figure 5.5)

6. select the Operating System (different versions of Ubuntu) to be used and the packages/repositories required (Figure 5.6)

7. set additional job properties like disk usage and memory usage (Figure 5.7)

After submitting the tasks, users should leave the nuBOINC Manager running and connected to the Internet in order to receive the results at any time.

Assuming users know how to invoke an application through the command line and which software are required, the process is straightforward, the nuBOINC Manager being the only new software users must interact with and having only 2 functions: create torrents and receive results. In conclusion, the

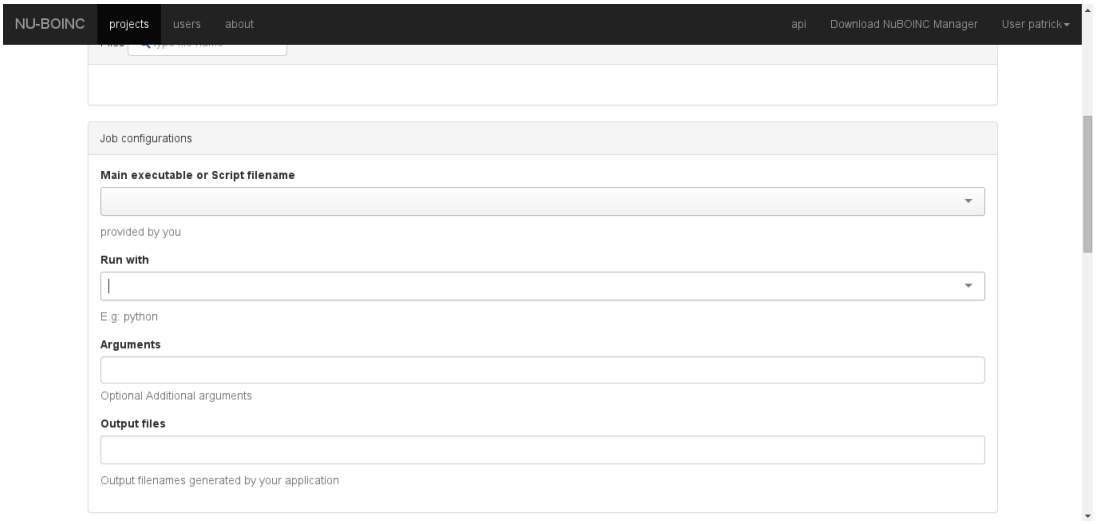


Figure 5.5: Job creation: job configuration

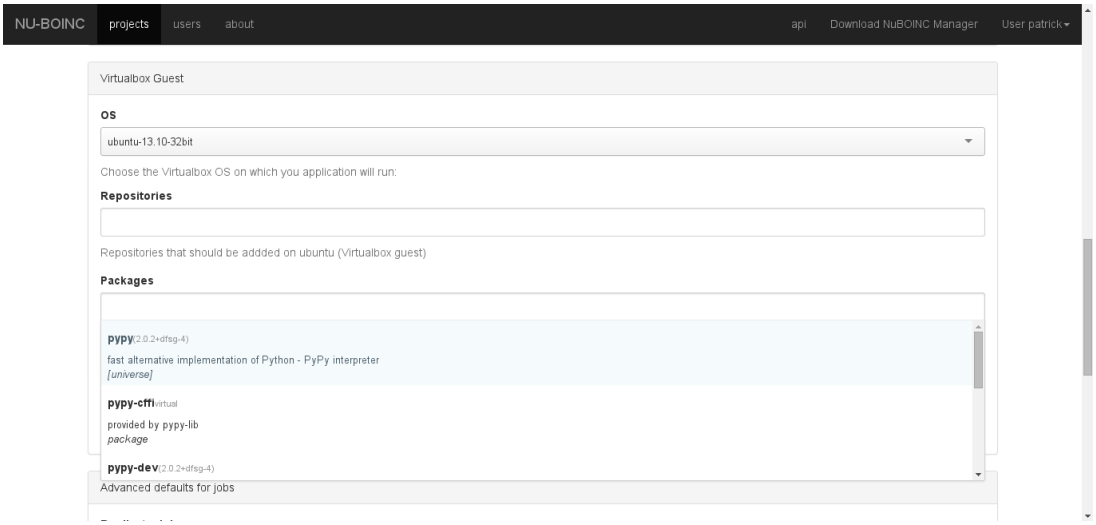


Figure 5.6: Job creation: virtual machine configuration

users can, after a small introduction, work quickly and independently.

5.2 Transfer Overhead reduction

It was also important to optimize the time consumed when downloading specific files. Files that are used in each subsequent task should only be downloaded once , if possible. The only part that can significantly increase the time to run a task is the download of the disk image and input files.

The disk image only needs to be downloaded once, therefore the time lost in the downloading process only applies to the first tasks: the one which is downloading it and the ones which are waiting for the download to finish. Subsequent tasks that run later do not need to download it again. The process needs to be repeated only when the temporary folder is cleaned. This applies for all tasks from different users.

Likewise, an input file may be downloaded once, but can be used by multiple tasks. This happens when one task downloads the input file and then another task also needs the same file. Since the files

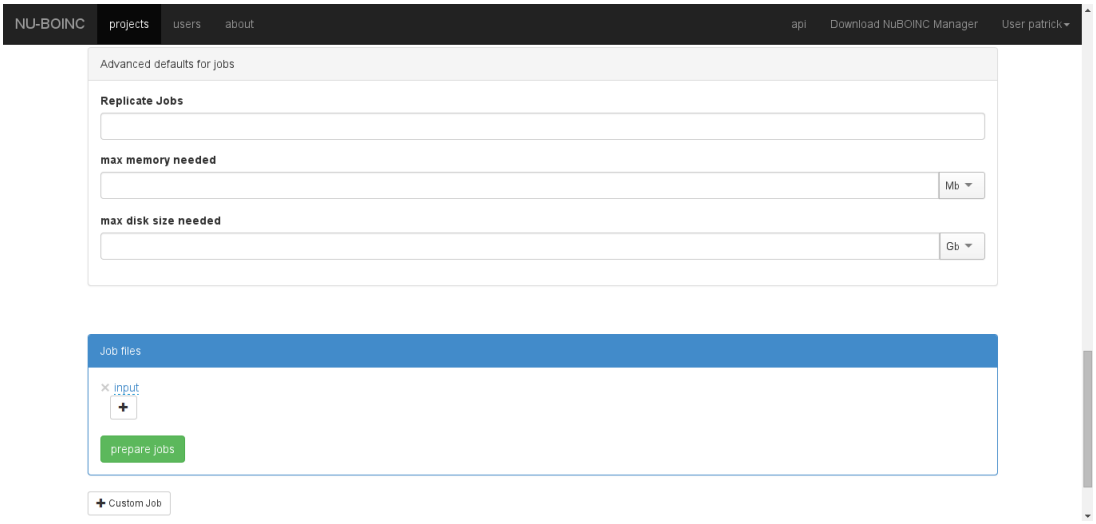


Figure 5.7: Job creation: set job memory/disk space requirements and input file names

are being seeded by the first task on the same computer, the second task can obtain it much faster, equivalent to a file copy operation.

5.3 Deployment

The nuBOINC server only has to be deployed once. Afterwards, since the infrastructure is public, everyone can use it. It is deployed in much the same way that the original BOINC server must be deployed. First, one needs to obtain the source code of the original BOINC server. Then our patch must be applied to the scheduler and afterwards it can be compiled. After following the BOINC installation guide, only a few more steps are needed to setup BOINC. One must copy the BOINC template files from our repository to the template directory and edit the BOINC projects configuration file (config.xml) to setup the following daemons:

- feeder –priority_order_plus_create_time 10 (or other number; see section 3.5)
- transitioner
- sample_trivial_validator -app nuboinc-32bit –sleep_interval 1 –no-credit
- sample_trivial_validator -app nuboinc-64bit –sleep_interval 1 –no-credit
- file_deleter (to delete the uploaded torrent files)
- trickle_credit –variety runtime –max_runtime 86400

Additionally, the following nuBOINC scripts must be configured and started (or added with full path to the configuration file): The work_generator and the server are compatible with PyPy which increases performance. The torrent script must be started with python.

- work_generator.py

- torrent.py
- server.py

The script ”server.py” should preferably be started with Gunicorn [91] and *-k gevent* as parameters. Gunicorn is also compatible with PyPy [92] A server configuration file example (see appendix A) is provided and should be renamed to config.yml as indicated. This file can then be adapted to:

- provide the right SQL access credentials
- specify the BOINC project path
- specify VirtualBox image torrent files
- specify the VirtualBox machines' credentials (all must use the same credentials)

The BOINC application is written in python and must be ”compiled” with *pyinstaller* [71] to create a single executable. By using this method, it is not needed to install python and other dependencies on a client's computer. Additionally, it is compatible with as many systems as on which python can run.

The BOINC application depends on PyQt5.2 [66] or higher, which only supports Python3, but the VirtualBox API only supports Python2. The same applies to PyInstaller. Therefore the PyQt code has to be compiled for Python2, which is actually supported, but unfortunately no binary distribution of PyQt exists for Python2.

To compile PyQt, a Qt installation [63] and SIP [93] are required. On Ubuntu those can be installed via the package manager. Additionally, the libtorrent python bindings [69] are needed, which can also be installed through the package manager.

After all dependencies are met, PyInstaller can be invoked in the source code directory with the parameter ”main.spec”, which is the name of the file that indicates PyInstaller what to do.

When the process is finished the file nuboinc.exe will appear in the *dist* directory and can be copied to the server BOINC Project.

To build those dependencies on windows we suggest to use visual studio, although we used mingw64 since we had it already installed. However, many problems aroused when compiling the Boost library, especially because of needed 64bit build and we were forced to install Visual Studio since Boost requires the executable ml64, which is included in visual studio.

There are also many problems concerning shared libraries that were not found and the only error message was ”A dynamic link library (DLL) initialization routine failed”, forcing me to install the ”Dependency Walker” program, which, as the name suggests, analyses a specific file for it's dependencies such as DLLs, displays them and verifies whether they exist or can be found.

The 64bit build is needed because VirtualBox API only allows the 64bit DLL file to be used on a 64bit system.

5.4 Experiments

In order to evaluate the usability, the server was deployed and the definition and execution of jobs where assessed. Three experiments with different input files, outputs and arguments where made. For each experiment, we created a torrent containing the input file names and possible executables or scripts. The first experiment performed consisted of a simple Python script reading one file, converting all characters to uppercase and writing them to an output file. In the following paragraphs we will explain each step taken. The first preparation step consisted in copying the python script and input files to one directory, for this experiment it will be called *test1*. The names of the various input files does not matter.

Next, the nuBOINC manager was used to generate the torrent file by selecting all files in the *test1* directory. As soon as it was generated the contained folder was opened, so as to display the location of the torrent file.

When creating jobs, the website will first require the generated torrent file. After uploading it the website will display the torrent files and provide options to select one file as executable, and to include files in each task. For *test1* the python script was selected as executable and python was selected as the executable's caller. After filling out the rest and triggering the create button the website will send queries to the server in order to create a job for each input file.

In order to test multiple input and output files, the previous application was adapted so that it applies the same process to two input files and writes the result to two output files. Furthermore, it was run with the Python interpreter PyPy, which demonstrates that the defined package PyPy was installed. In the last experiment, a video should be processed with avconc to add an effect. First, the video was split into 5 parts using avconv. Afterwards, it was possible to apply the same steps as done before. On the website, we defined two packages that are required by avconv (libav-tools and libavcodex-extra-53), note that this package combinations is valid for Ubuntu 13.10, but not for Ubuntu 14.04. Consequently, it is important to verify the package combination on the target Ubuntu version. All tasks were executed successfully and all results were transmitted back to the nuBOINC Manager.

5.4.1 Credits

We performed 3 experiments to test the slightly modified credit system. In order to perform the experiment we registered 3 users: User1, User2 and User3.

5.4.1.A Experiment One: Granting Credit

The objective of this experiment is to verify that jobs created by one user and executed by the same are not granted credit. For the purpose of this experiment the accounts User1 and User2 are used. Both submit jobs, but only User1 will process jobs. First we only submitted jobs using User1.

The submitted jobs are the same previously used to process video segments with avconv. The data in the following tables have been obtained from the BOINC database. Before executing the jobs, each user had 0 credits, afterwards they are still 0, because User1 processed his own jobs. Note that the

credits are granted using trickle messages. Normally, this is done by the validator, which grants credits to results and workunits. In this case no credits are granted to the results and workunits but instead are applied directly to the user and the host. The trickle message is only sent when the job is processed by any other user than the project creator.

Total Credits	Before processing	After processing
User1	0	0
User2	0	0

Table 5.1: Credits after processing jobs from User1

Secondly, we submitted the same jobs with User2, while User1 still processes all jobs. In this case User1 receives credits since he processed jobs from another user.

Total Credits	Before processing	After processing
User1	0	19.41
User2	0	0

Table 5.2: Credits after processing jobs from User2

5.4.1.B Experiment Two: Prioritization

The objective of this experiment is to verify that jobs from users with higher priority are indeed prioritized. To test the priority, we use all three accounts: User1, User2 and User3. User1 will process the jobs, whereas User2 and User3 will submit jobs. For this experiment we assigned a random total credit of 48.89 to User2 and set the servers maximum credit limit to 100. User3 remains with zero credit. First, we submit jobs with User3 and then with User2. With both Users we submit 20 jobs.
The jobs have the following order and priority in the database:

user	#job	priority
user3	1	0
user3	2	0
user3	3	0
user3	4	0
...		0
user3	10	0
user2	11	91
user2	12	91
user2	13	91
user2	14	91
...		91
user2	20	91

Table 5.3: Created jobs database order and priority

Using the formula 3.2, we can check if the priority 91 is correct: For this experiment we used:
maximum credit(tau): 100

users total credit: 48.89

result: $round(100 * (1 - e^{-48.89/100*5})) = round(91.323) = 91$
The result is rounded because the database and BOINC internal code only accepts Integers. When the BOINC client on User1 request jobs, it will receive the jobs ordered by their priority. In this case, the Client received 10 jobs, all of User2, even though User3 submitted first.

5.4.1.C Experiment Three: Prioritization with elapsed time

This last experiment verifies the correct prioritization of tasks that have been created at an earlier time. Enough time must have passed, such that they are ranked higher than newly submitted jobs with a higher initial priority. Refer to Chapter 3 *Scheduler and Feeder*. Three users will be required to perform this experiment: User1, User2, User3. In particular User1 will begin with a priority of 50, whereas User2 will begin with zero priority. Initially no user is processing jobs.

For the sake of this experiment we reduced the time it takes for the priority to increases in 10 minutes. To clarify, the priority should exceed the value 50 in 10 minutes.

As explained in Chapter 3 it is possible to set the time it takes for a priority of zero to achieve a value of 100. The time is given in number of days and the priority increases linearly. Conclusively the time must be set to 20 minutes to achieve the value 50 in 10 minutes. This is equivalent to 0.01388 days. $0.01388 = 20/24/60$

First, User2 will submit 20 tasks. After more than 10 minutes, User1 will submit another 20 tasks. At a later time, User3 will begin processing jobs.

After User3 requests Jobs, it can be observed that only Jobs from User2 where downloaded. This proofs that jobs from User2 were prioritized, even though User1 has a higher priority.

We did a second test with only 5 jobs from each user. In that case, jobs from both users were downloaded. The correct prioritization could still be observed through the BOINC client's Log file and by the fact that the jobs from User2 started first.

5.4.2 Conclusion

In conclusion, all experiments yield the expected results:

- Users will first process they own jobs, but not receive any credit.
- Users with many credits receive a higher prioritization for their submitted jobs and users with scarce credits receive a low prioritization for they submitted jobs.
- The priority of jobs increase over time .

6

Conclusions and Future Work

Contents

6.1 Conclusion 56

6.2 Future Work 56

6.1 Conclusion

We managed to extend the original BOINC infrastructure into a system that allows users to submit jobs. For this purpose, a job submission user interface was added to BOINC, allowing the definition of each job's input files and parameters without any programming knowledge. The jobs can also be configured individually using the REST API, allowing institutes or advanced users to setup their own work generator to submit many different jobs. Both server and the website have been improved to provide a better user experience by using modern javascript and CSS frameworks. Additionally, they were developed with current conventions in mind.

A special BOINC application was developed that downloads all required files using the BitTorrent protocol and then executes the defined processing software in a secure environment.

All job files from one project are provided by one torrent file, even when each job requires different input files. The torrent file should be generated and be seed by a software we provide: the nuBOINC Manager. Our software generates the torrent in such way, that when downloaded, it only allocates the required disk space for the files used by the job.

Except the torrent file, no other files are saved on the server. As such, the output files are also directly sent to the project creator. The nuBOINC Manager is responsible for receiving the output files and saving them in an organized structure, so that they can be processed in an autonomous way.

By allowing the execution of any application that can run on Linux, users do not have to develop the BOINC applications that will be executed on remote computers or do not need to deal with BOINC integration when developing their own custom software.

However, it is impossible to determine if the submitted applications are secure. Consequently, they run in a secure virtual machine, protecting the host computer from malicious code. Additionally, the network connection will be removed from the machine as soon as it's provision completes, including the install of packages. This prevents malicious code to access the local network. The virtual machine will be provided by VirtualBox, which can be downloaded together with the BOINC client.

To guarantee some fairness, the way scheduling is performed has been modified on the server. In the event that a project creator has a computer connected to nuBOINC and has jobs to be executed, the scheduler sends his jobs first. Moreover a prioritizing system was implemented that prefers jobs from users with a high amount of credits or prefers jobs that were created at an earlier time.

Finally, experiments have been performed to guarantee the correct implementation of file transfer, job submission, job execution, software installation , credit granting and job prioritization. All of them yield the expected results.

6.2 Future Work

A drawback of custom processing software is that it is impossible to determine how long the application will run and thus no proper progress can be shown in the BOINC Client. This could cause users to be unsure if the job is still executing or some unknown error occurred. This should be improved

by allowing users to set a mean value of the application's execution time or the approximate FLOPs required.

Additionally, the credit system should be improved so that no credit reset is required and users pay with credits after a job has been processed.

It is important to realize that all developed programs are still work in progress. Many edge cases are not handled, even an Internet disconnection may cause the program to stop during the preparation step. Moreover the nuBOINC applications does not support resuming and also waits forever to receive and send files. Additionally, the credits are granted by using the trickle systems which does not give credits in a fair way and therefore the Boinc Client should be extended to send files directly to the project creator. Then again, this could prevent current BOINC users to join nuBOINC because they have to download a special client.

The time consumed for installing the necessary packages can be further improved by making snapshots after a software was installed, so as to reuse the software without the need to download and install it again and thus save time. Additionally, software to provide more complex setups can be used. Such as Chef or Puppet to install and manage software in a safe way [94]. These tools are generally used to setup complex servers and applications in the cloud, nevertheless, they can also be used for this simple tasks.

Finally, the server's security could be improved by using HMAC [95]. In general, it is important to further investigate the security of this infrastructure.

Bibliography

[1] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop, 2008. GCE'08*. Ieee, 2008, pp. 1–10.

[2] D. P. Anderson and G. Fedak, "The computational and storage potential of volunteer computing," in *Sixth IEEE International Symposium on Cluster Computing and the Grid, 2006. CCGRID 06.*, vol. 1. IEEE, 2006, pp. 73–80.

[3] (2014) Folding home. [Online]. Available: http://folding.stanford.edu/

[4] U. of California. (2014) Project list. [Online]. Available: http://boinc.berkeley.edu/wiki/project_list

[5] (2014) Boincstats. [Online]. Available: http://boincstats.com/

[6] (2014) Fightmalaria@home. [Online]. Available: http://boinc.ucd.ie/fmah/

[7] (2014) Dna@home. [Online]. Available: http://volunteer.cs.und.edu/dna/

[8] U. of California. (2013) Account managers. [Online]. Available: http://boinc.berkeley.edu/wiki/account_managers

[9] P. P. Cano and M. Vargas-Lombardo, "Security threats in volunteer computing environments using the berkeley open infrastructure for network computing (boinc)." *International Journal of Computer Technology & Applications*, vol. 3, no. 3, 2012.

[10] U. of California. (2013) Boinc security. [Online]. Available: http://boinc.berkeley.edu/wiki/BOINC_Security

[11] ——. (2013) Boinc security issues. [Online]. Available: http://boinc.berkeley.edu/trac/wiki/SecurityIssues

[12] D. P. Anderson *et al.* (2014) The boinc wrapper. [Online]. Available: http://boinc.berkeley.edu/trac/wiki/WrapperApp

[13] U. of California. (2010) Validation and replication. [Online]. Available: http://boinc.berkeley.edu/trac/wiki/ValidationSummary

[14] D. P. Anderson *et al.* Handling completed jobs. [Online]. Available: http://boinc.berkeley.edu/trac/wiki/AssimilateIntro

[15] (2014) big ugly rendering project. [Online]. Available: http://burp.renderfarming.net/

[16] U. of Leiden. (2014) Leiden classical. [Online]. Available: http://boinc.gorlaeus.net/

[17] B. Foundation. (2014) Big buck bunny. [Online]. Available: http://bbb3d.renderfarming.net/

[18] Leiden. Leiden classical-builder. [Online]. Available: http://boinc.gorlaeus.net/ClassicalBuilder.php

[19] S. Choi, H. Kim, E. Byun, M. Baik, S. Kim, C. Park, and C. Hwang, "Characterizing and classifying desktop grid," in *Seventh IEEE International Symposium on Cluster Computing and the Grid, 2007. CCGRID 2007.* IEEE, 2007, pp. 743–748.

[20] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor-a hunter of idle workstations," in *8th International Conference on Distributed Computing Systems, 1988.* IEEE, 1988, pp. 104–111.

[21] V. Sekhri, R. Gupta, and A. K. Somani, "Compup2p: A light-weight architecture for internet computing," in *2nd International Conference on Broadband Networks. BroadNets 2005.* IEEE, 2005, pp. 866–875.

[22] N. Nisan, S. London, O. Regev, and N. Camiel, "Globally distributed computation over the internet-the popcorn project," in *18th International Conference on Distributed Computing Systems, 1998. Proceedings.* IEEE, 1998, pp. 592–601.

[23] R. Mason and W. Kelly, "G2-p2p: a fully decentralised fault-tolerant cycle-stealing framework," in *Proceedings of the 2005 Australasian workshop on Grid computing and e-research-Volume 44*. Australian Computer Society, Inc., 2005, pp. 33–39.

[24] N. Andrade, W. Cirne, F. Brasileiro, and P. Roisenberg, "Ourgrid: An approach to easily assemble grids with equitable resource sharing," in *Job scheduling strategies for parallel processing*. Springer, 2003, pp. 61–86.

[25] Ourgrid. [Online]. Available: http://ourgrid.org/

[26] L. Wang, J. Tao, M. Kunze, A. C. Castellanos, D. Kramer, and W. Karl, "Scientific cloud computing: Early definition and experience." in *HPCC*, vol. 8, 2008, pp. 825–830.

[27] C. Vecchiola, S. Pandey, and R. Buyya, "High-performance cloud computing: A view of scientific applications," in *2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks (ISPAN)*. IEEE, 2009, pp. 4–16.

[28] Amazon. High performance computing. [Online]. Available: https://aws.amazon.com/hpc/

[29] M. R. Rahman, "A survey of incentive mechanisms in peer-to-peer systems," University of Waterloo, Tech. Rep., 2009.

[30] B. Cohen. (2012) The bittorrent protocol specification. [Online]. Available: http://www.bittorrent.org/beps/bep_0003.html

[31] ——, “Incentives build robustness in bittorrent,” in *Workshop on Economics of Peer-to-Peer systems*, vol. 6, 2003, pp. 68–72.

[32] R. Rahman, M. Meulpolder, D. Hales, J. Pouwelse, D. Epema, and H. Sips, “Improving efficiency and fairness in p2p systems with effort-based incentives,” pp. 1–5, 2010.

[33] D. Vega, R. Meseguer, F. Freitag, and S. F. Ochoa, “Effort-based incentives for resource sharing in collaborative volunteer applications,” pp. 37–42, 2013.

[34] U. of California. (2011) Alternative mechanisms for granting credit. [Online]. Available: http://boinc.berkeley.edu/trac/wiki/CreditAlt

[35] ——. (2012) A new system for runtime estimation and credit. [Online]. Available: http://boinc.berkeley.edu/trac/wiki/CreditNew

[36] P. D. Rodrigues, C. Ribeiro, and L. Veiga, “Incentive mechanisms in peer-to-peer networks,” in *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*. IEEE, 2010, pp. 1–8.

[37] T.-Y. Wu, W.-T. Lee, N. Guizani, and T.-M. Wang, “Incentive mechanism for p2p file sharing based on social network and game theory,” *Journal of Network and Computer Applications*, vol. 41, pp. 47–55, 2014.

[38] D. P. Anderson *et al.* Increasing server capacity. [Online]. Available: http://boinc.berkeley.edu/trac/wiki/MultiHost

[39] D. P. Anderson, E. Korpela, and R. Walton, “High-performance task distribution for volunteer computing,” in *First International Conference on e-Science and Grid Computing, 2005.* IEEE, 2005, pp. 8–pp.

[40] F. Costa, L. Silva, G. Fedak, and I. Kelley, “Optimizing data distribution in desktop grid platforms,” *Parallel Processing Letters*, vol. 18, no. 03, pp. 391–410, 2008.

[41] F. Costa, L. Silva, I. Kelley, and I. Taylor, “Peer-to-peer techniques for data distribution in desktop grid computing platforms,” in *Making Grids Work*. Springer, 2008, pp. 377–391.

[42] F. Costa, L. Silva, G. Fedak, and I. Kelley, “Optimizing the data distribution layer of boinc with bittorrent,” pp. 1–8, 2008.

[43] A. Bakker. (2009) Merkle torrent. [Online]. Available: http://bittorrent.org/beps/bep_0030.html

[44] U. Forum. (2008) Universal plug and play (networking protocols). [Online]. Available: http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0.pdf

[45] O. Corporation. (2014) Virtualbox. [Online]. Available: https://www.virtualbox.org/

[46] (2014) Vmware. [Online]. Available: http://www.vmware.com/

[47] (2014) Kernel based virtual machine. [Online]. Available: http://www.linux-kvm.org/

[48] (2014) Xen. [Online]. Available: http://www.xenproject.org/

[49] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," vol. 41, no. 3, pp. 275–287, 2007.

[50] dotCloud. (2014) Docker. [Online]. Available: https://www.docker.com/

[51] M. Bolte, M. Sievers, G. Birkenheuer, O. Niehörster, and A. Brinkmann, "Non-intrusive virtualization management using libvirt," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2010, pp. 574–579.

[52] O. Corporation, "Virtualbox user manual - special image write modes," 2014.

[53] P. Ferrie, "Attacks on virtual machine emulators," in *AVAR Conference, Auckland, Symantec Advanced Threat Research*, 2006.

[54] D. Hyde. (2009) A survey on the security of virtual machines. [Online]. Available: http://www.cse.wustl.edu/~jain/cse571-09/ftp/vmsec/index.html

[55] J. S. Reuben, "A survey on virtual machine security," *Helsinki University of Technology*, 2007.

[56] (2012) Oracle vm virtualbox two vulnerabilities. [Online]. Available: https://secunia.com/advisories/47626/

[57] D. P. Anderson *et al.* job. [Online]. Available: http://boinc.berkeley.edu/trac/wiki/JobIn

[58] ——. Input and output templates. [Online]. Available: http://boinc.berkeley.edu/trac/wiki/JobTemplates

[59] ——. Trickle messages. [Online]. Available: http://boinc.berkeley.edu/trac/wiki/TrickleMessages

[60] O. Corporation, "Virtualbox user manual - supported guest operating systems," 2014.

[61] U. of California. (2013) Startup and status data. [Online]. Available: http://boinc.berkeley.edu/trac/wiki/StatusApi

[62] ——. (2013) Computational credit. [Online]. Available: http://boinc.berkeley.edu/wiki/Computation_credit

[63] Digia. (2014) Qt. [Online]. Available: http://qt-project.org/

[64] wxwidgets. [Online]. Available: http://www.wxwidgets.org/

[65] M. Dorman. A pythonic implementation of the entire virtualbox main api. [Online]. Available: https://github.com/mjdorma/pyvbox

[66] Riverbank. (2014) Pyqt. [Online]. Available: http://www.riverbankcomputing.co.uk/software/pyqt/intro

[67] G. Bajo, M. Zibricky, and H. Goebel. (2014) Pyside. [Online]. Available: http://qt-project.org/wiki/PySide

[68] B. Cohen. Bittorrent. [Online]. Available: http://sourceforge.net/projects/bittorrent/

[69] A. Norberg. (2014) The opensource libtorrent library. [Online]. Available: http://libtorrent.org/

[70] BitTorrent. (2014) Bittorrent sync. [Online]. Available: http://www.getsync.com/

[71] M. Zibricky, H. Goebel, and G. Bajo. (2014) Pyinstaller. [Online]. Available: http://www.pyinstaller.org/

[72] Y. Katz, T. Dale, P. Wagenet *et al.* (2014) Ember.js: A framework for creating ambitious web applications. [Online]. Available: http://emberjs.com/

[73] Google. Angularjs. [Online]. Available: https://angularjs.org/

[74] DocumentCloud. Backbonejs. [Online]. Available: http://backbonejs.org/

[75] Gruntjs. [Online]. Available: http://gruntjs.com/

[76] A. Shakarchi. Bootrstrap for ember. [Online]. Available: https://github.com/ember-addons/bootstrap-for-ember

[77] Phalcon. [Online]. Available: http://phalconphp.com/

[78] Yaf. [Online]. Available: http://yafdev.com/

[79] Propel. [Online]. Available: http://propelorm.org/

[80] Pypy. [Online]. Available: http://pypy.org/

[81] M. Hellkamp. Bottle: Python web framework. [Online]. Available: http://bottlepy.org/

[82] A. Ronacher. Flask. [Online]. Available: http://flask.pocoo.org/

[83] M. Bayer. Sqlalchemy. [Online]. Available: http://www.sqlalchemy.org/

[84] D. Bilenko *et al.* (2014) Gevent. [Online]. Available: http://www.gevent.org/

[85] D. Crockford, “The application/json media type for javascript object notation (json),” 2006.

[86] O. Ben-Kiki, C. Evans, and B. Ingerson, “Yaml ain’t markup language version 1.1,” *Working Draft 2008*, vol. 5, p. 11, 2001.

[87] BitComet. (2010, November) Align file to piece boundary. [Online]. Available: http://wiki.bitcomet.com/align_file_to_piece_boundary

[88] screen - linux man page. [Online]. Available: http://linux.die.net/man/1/screen

[89] I. Naoki *et al.* (2014) Pymysql: Pure-python mysql client. [Online]. Available: https://github.com/PyMySQL/PyMySQL

[90] M. Otto, J. Thornton *et al.* (2014) Twitter bootstrap. [Online]. Available: http://getbootstrap.com/

[91] B. Chesneau. (2014) Gunicorn: a python wsgi http server for unix. [Online]. Available: http://gunicorn.org/

[92] (2014) Gunicorn pypy compatibility. [Online]. Available: https://bitbucket.org/pypy/compatibility/wiki/gunicorn

[93] Riverbank. (2014) Sip. [Online]. Available: http://www.riverbankcomputing.com/software/sip/intro

[94] F. Önnberg, "Software configuration management : A comparison of chef, cfengine and puppet," p. 33, 2012.

[95] H. Krawczyk, R. Canetti, and M. Bellare, "Hmac: Keyed-hashing for message authentication," 1997.

A

Server Config File

```yaml
#rename to config.yml
sql:
  host: ~
  port: ~
  username: root
  password: root
  database: test

boinc:
  path: projects/test

server:
  api: http://ubuntu.local/api
  maxcredit: 200000
  time-period: 100 # days
  port: 8080

torrent:
  tracker: udp://ubuntu.local:6969/announce
  seeds:
    -
      torrent: nuboinc.torrent
      data-dir: data


virtualbox:
  login:
    username: ubuntu
    password: reverse
  torrent-url: http://ubuntu.local/nuboinc/data/nuboincvbox.vmdk.zip.torrent
```

B

Job Info File

```
{
    "project": {
        "user_id": "1",
        "name": "test",
        "jobid": 331,
        "user": "patrick",
        "api-server": "http://ubuntu.local/api/"
    },
    "application": {
                "outputs": ["out.avi"],
                "app": null,
                "with": "avconv",
                "arguments": "-i input.avi -b:a 192k -vf negate -codec:v
                    libx264 -crf 18 -preset slow -pix_fmt yuv420p  out.
                   avi",
                "inputs": [
                        {
                                "localname": "input.avi",
                                "remotename": "part2.avi"
                        }
                ]
        },
        "advanced": {
                "rsc_memory_bound": 104857600,
                "rsc_disk_bound": 536870912
        },
        "virtualbox": {
                "machine": {
                        "name": "ubuntu-13.10-32bit",
                        "link": "http://ubuntu.local/nuboinc/data/
                           nuboincvbox.vmdk.torrent",
                        "architecture": "32bit",
                        "ostype": "Linux_64",
                        "codename": "saucy",
                        "os": "ubuntu"
                },
                "packages-needed": [
                        "libav-tools",
```

```
                        "libavcodec-extra-53"
                ],
                "login": {
                        "username": "ubuntu",
                        "password": "reverse"
                },
                "repositories": [],
                "pkgfiles": []
        }
}
```

C

Server Changes

```
diff --git a/sched/feeder.cpp b/sched/feeder.cpp
index 2bd61c2..2e9045a 100644
--- a/sched/feeder.cpp
+++ b/sched/feeder.cpp
@@ -709,6 +709,7 @@ void usage(char *name) {
         "  [ --priority_asc ]                order by increasing \"priority
            \" field of result\n"
         "  [ --priority_order ]              order by decreasing \"priority
            \" field of result\n"
         "  [ --priority_order_create_time ] order by priority, then by
            increasing WU create time\n"
+        "  [ --priority_order_plus_create_time X] nuBOINC: order by
    priority plus create time diff to now. X days until 100 is reached from
    0\n"
         "  [ --purge_stale x ]               remove work items from the
            shared memory segment after x secs\n"
         "                                    that have been there for
            longer then x minutes\n"
         "                                    but haven't been assigned\n"
@@ -747,6 +748,24 @@ int main(int argc, char** argv) {
             order_clause = "order by r1.priority desc ";
         } else if (is_arg(argv[i], "priority_order_create_time")) {
             order_clause = "order by r1.priority desc, r1.workunitid";
+        } else if (is_arg(argv[i], "priority_order_plus_create_time")) {
+            if (argc < i+1){
+                log_messages.printf(MSG_CRITICAL, "%s requires an argument
    \n\n", argv[--i]);
+                usage(argv[0]);
+                exit(1);
+            }
+            char *p;
+            float converted = strtof(argv[++i], &p);
+            if(*p){
+                log_messages.printf(MSG_CRITICAL, "%s requires a number\n\
    n", argv[--i]);
+                usage(argv[0]);
+                exit(1);
+            }
+            log_messages.printf(MSG_DEBUG, "running with createtime weight
```

```
    %f\n", converted);
+           sprintf(order_buf,"order by r1.priority + (UNIX_TIMESTAMP()-r1
    .create_time)/86400.0 * %f desc ",
+                   converted
+           );
+           order_clause = order_buf;
        } else if (is_arg(argv[i], "by_batch")) {
            // Evenly distribute work among batches
            // The 0=1 causes anything before the union statement
diff --git a/sched/sched_array.cpp b/sched/sched_array.cpp
index c7344c9..308aaee 100644
--- a/sched/sched_array.cpp
+++ b/sched/sched_array.cpp
@@ -357,6 +357,15 @@ void send_work_old() {
     g_wreq->user_apps_only = true;
     g_wreq->infeasible_only = false;

+    g_wreq->userjobs_only = true;
+    if (config.debug_array) {
+        log_messages.printf(MSG_NORMAL,
+            "[array] scanning for user jobs\n"
+        );
+    }
+    if (scan_work_array()) return;
+    g_wreq->userjobs_only = false;
+
     // give top priority to results that require a 'reliable host'
     //
     if (g_wreq->has_reliable_version) {
diff --git a/sched/sched_check.cpp b/sched/sched_check.cpp
index 43d6f35..15e8b2c 100644
--- a/sched/sched_check.cpp
+++ b/sched/sched_check.cpp
@@ -438,6 +438,18 @@ int slow_check(
        }
    }

+   //nuboinc: send userjobs first to user
+   if(g_wreq->userjobs_only){
```

```
+        DB_BASE base("",&boinc_db);
+        int user_id = -1;
+        char query[255]="";
+        sprintf(query, "SELECT user_id FROM user_project,project_job WHERE
    project_job.user_project_id=user_project.id AND project_job.workunit_id
   =%d", wu.id);
+        retval = base.get_integer(query, user_id);
+        if(retval || user_id != g_reply->user.id){
+            return 1;
+        }
+    }
+
     // Checks that require looking up the WU.
     // Lump these together so we only do 1 lookup
     //
diff --git a/sched/sched_types.h b/sched/sched_types.h
index 0fc2e3b..8a86fc2 100644
--- a/sched/sched_types.h
+++ b/sched/sched_types.h
@@ -363,6 +363,7 @@ struct WORK_REQ_BASE {
     bool user_apps_only;
     bool beta_only;
     bool locality_sched_lite;
+    bool userjobs_only;
         // for LSL apps, send only jobs where client has > 0 files

     bool resend_lost_results;
diff --git a/sched/trickle_credit.cpp b/sched/trickle_credit.cpp
index a98324a..6f0f545 100644
--- a/sched/trickle_credit.cpp
+++ b/sched/trickle_credit.cpp
@@ -40,9 +40,9 @@
 #include "sched_msgs.h"
 #include "trickle_handler.h"

-double flops_50_percentile;
+double flops_50_percentile = 1000;
     // default if host value is <= 0
-double flops_95_percentile;
```

```
+double flops_95_percentile = 2000;
     // limit for cheat-proofing
 double max_runtime = 0;

@@ -63,14 +63,14 @@ int handle_trickle_init(int argc, char** argv) {

     DB_HOST host;
     retval = host.fpops_percentile(50, flops_50_percentile);
-    if (retval) {
+    if (retval < 0 && retval != ERR_NULL) {
         log_messages.printf(MSG_CRITICAL, "fpops_percentile failed: %d\n",
             retval);
-        return retval;
+        return 0;
     }
     retval = host.fpops_percentile(95, flops_95_percentile);
-    if (retval) {
+    if (retval < 0 && retval != ERR_NULL) {
         log_messages.printf(MSG_CRITICAL, "fpops_percentile failed: %d\n",
             retval);
-        return retval;
+        return 0;
     }

     log_messages.printf(MSG_NORMAL, "default FLOPS: %f\n",
        flops_50_percentile);
```